Lecture Notes in Computer Science 1872
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Foreword

In 1996 the International Federation for Information Processing (IFIP) established its first Technical Committee on foundations of computer science, TC1. The aim of IFIP TC1 is to support the development of theoretical computer science as a fundamental science and to promote the exploration of fundamental concepts, models, theories, and formal systems in order to understand laws, limits, and possibilities of information processing.

This volume constitutes the proceedings of the first IFIP International Conference on Theoretical Computer Science (IFIP TCS 2000) – Exploring New Frontiers of Theoretical Informatics – organized by IFIP TC1, held at Tohoku University, Sendai, Japan in August 2000.

The IFIP TCS 2000 technical program consists of invited talks, contributed talks, and a panel discussion. In conjunction with this program there are two special open lectures by Professors Jan van Leeuwen and Peter D. Mosses.

The decision to hold this conference was made by IFIP TC1 in August 1998, and since then IFIP TCS 2000 has benefited from the efforts of many people; in particular, the TC1 members and the members of the Steering Committee, the Program Committee, and the Organizing Committee of the conference. Our special thanks go to the Program Committee Co-chairs:

Track (1): Jan van Leeuwen (U. Utrecht), Osamu Watanabe (Tokyo Inst. Tech.)
Track (2): Masami Hagiya (U. Tokyo), Peter D. Mosses (U. Aarhus).

The details of the conference were planned by the Steering Committee with the help of the PC Co-chairs. The Steering Committee members are

Giorgio Ausiello (U. Roma "La Sapienza") *Chair*
Wilfried Brauer (TU München)
Takayasu Ito (Tohoku U.)
Michael O. Rabin (Harvard U.)
John Staples (U. Queensland)
Joseph Traub (Columbia U.)

Professor Michael O. Rabin of Harvard University accepted our invitation to be the banquet speaker of IFIP TCS 2000.

IFIP TCS 2000 and the IFIP Technical Committee TC1 gratefully acknowledge the partial support provided by Tohoku University and Sendai Tourism & Convention Bureau, and the cooperation of the following organizations:

Information Processing Society of Japan
Japan Society of Software Science and Technology
European Association of Theoretical Computer Science
Association of Symbolic Logic
Association for Computing Machinery-SIGACT

We would like to thank Plamen Nedkov and Dorothy Hayden of the IFIP office for their assistance on IFIP procedures, and we also thank Alfred Hoffman of Springer-Verlag for his assistance in the publication of the proceedings. Masahiko Ohtomo, Shinya Miyakawa, Nobuto Izumi, and Takuya Ohishi of Tohoku University lent their assistance in making local arrangements and in preparing the proceedings and the conference web pages. Finally, we would like to express our sincere thanks to all those who helped organize, and participated in, the IFIP TCS 2000 conference for their invaluable contributions.

May 2000

Giorgio Ausiello
Takayasu Ito
Conference Co-chairs

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Jan van Leeuwen Osamu Watanabe
Masami Hagiya Peter D. Mosses
Takayasu Ito (Eds.)

Theoretical Computer Science

Exploring New Frontiers of Theoretical Informatics

International Conference IFIP TCS 2000
Sendai, Japan, August 17-19, 2000
Proceedings

Springer

Series Editors
Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors
Jan van Leeuwen
University of Utrecht, Department of Computer Science
Centrumgebouw Noord, Office A309
Padualaan 14, De Uithof, Utrecht, The Netherlands
E-mail: jan@cs.uu.nl

Osamu Watanabe
Tokyo Institute of Technology, Department of Information Science
Ookayama 2-12-1, Meguro-ku, Tokyo 152-8552, Japan
E-mail: watanabe@is.titech.ac.jp

Masami Hagiya
The University of Tokyo, Graduate School of Science
Department of Information Science, Bunkyo-ku, Tokyo 113-0033, Japan
E-mail:hagiya@is.s.u-tokyo.ac.jp

Peter D. Mosses
University of Aarhus, Department of Computer Science
Ny Munkegarde, Bldg. 540, 8000 Aarhus C, Denmark
E-mail: pdmosses@daimi.au.dk

Takayasu Ito
Tohoku University, Graduate School of Information Sciences
Department of Computer and Mathematical Sciences, Sendai, Japan 980-8579
E-mail: ito@ito.ecei.tohoku.ac.jp

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme
Theoretical computer science : exploring new frontiers of theoretical
informatics ; international conference ; proceedings / IFIP TCS 2000,
Sendai, Japan, August 17 - 19, 2000. J. van Leeuwen ... (ed.). -
Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1872)
ISBN 3-540-67823-9

CR Subject Classification (1998): F, D.3, E.1, I.3, C.2

ISSN 0302-9743
ISBN 3-540-67823-9 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH
© Springer-Verlag Berlin Heidelberg 2000
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10722337 06/3142 5 4 3 2 1 0

Preface

IFIP TCS 2000 is the first international conference organized by IFIP TC1, whose activities cover the entire field of theoretical computer science. Reflecting the current activities in theoretical computer science the major topics of the conference were chosen, forming the two tracks:

Track (1) on Algorithms, Complexity, and Models of Computation,
Track (2) on Logic, Semantics, Specification, and Verification.

The program of IFIP TCS 2000 included the presentations of eighteen contributed papers of Track (1) and fourteen contributed papers of Track (2). The Program Committee selected them from forty submissions to Track (1) and thirty submissions to Track (2), with the help of additional referees.

The invited speakers consist of the three keynote plenary invited speakers, and the three invited speakers for each track; they were chosen by the Steering Committee and the PC Co-chairs.

This volume constitutes the record of the technical program, consisting of the contributed papers and the invited talks. We had the pleasure of chairing the program committee of the first IFIP International Conference on Theoretical Computer Science with collaboration of the conference chairs. We are extremely grateful to Takayasu Ito and his staff, who helped us in making and announcing the call for papers, the program, and their web pages, and in preparing the proceedings.

We would like to express our thanks to the other members of the Program Committee and the additional referees, who are listed below, for their help in reviewing all submissions and for selecting the papers.

May 2000

Jan van Leeuwen
Osamu Watanabe
Masami Hagiya
Peter D. Mosses
Co-chairs

Program Committee

Track (1) on Algorithms, Complexity, and Models of Computation

Ricardo Baeza-Yates (U. Chile)
Siu-Wing Cheng (Hong Kong U. Science & Tech.)
Felipe Cucker (City U. Hong Kong)
Rosario Gennaro (IBM T.J. Watson Research)
Alan Gibbons (U. Liverpool)
Andrew V. Goldberg (InterTrust STAR Lab, USA)
Ernst Mayr (TU München)
Hiroshi Nagamochi (Kyoto U.)
Kouichi Sakurai (Kyushu U.)
Jan van Leeuwen (U. Utrecht)
Paul Vitanyi (CWI, Amsterdam)
Osamu Watanabe (Tokyo Inst. Tech.)
Jiří Wiedermann (Academy of Sciences, Prague)
Takashi Yokomori (Waseda U.)

Track (2) on Logic, Semantics, Specification, and Verification

Samson Abramsky (U. Edinburgh)
Egidio Astesiano (U. Genova)
Luca Cardelli (Microsoft, Cambridge)
Javier Esparza (TU München)
Masami Hagiya (U. Tokyo)
Naoki Kobayashi (U. Tokyo)
Peter D. Mosses (U. Aarhus)
Benjamin Pierce (U. Pennsylvania)
Davide Sangiorgi (INRIA, Sophia Antipolis)
John Staples (U. Queensland)
Andrzej Tarlecki (Warsaw U.)
P. S. Thiagarajan (Chennai Math. Inst., India)
Kazunori Ueda (Waseda U.)

Additional Referees

Track (1) on Algorithms, Complexity, and Models of Computation

Hee-Kap Ahn
Paul Dunne
Toshihiro Fujito
Leszek Gasieniec
Mordecai Golin
Alexander Hall
Toru Hasunuma
Alejandro Hevia
Klaus Holzapfel
Costas S. Iliopoulos
Marcel Jirina Jr.
Satoshi Kobayashi
Petr Kurka
Kazuhisa Makino
Eiji Miyano
Ken-ichi Morita
Martin Raab
Wolfciech Rytter
Yasubumi Sakakibara
Peter Savicky
Mark Scharbrodt
Jack Snoeyink
Sei-ichi Tani
Takeaki Uno
Ulrich Voll
Ronald de Wolf
Deng Xiao-Tie
Yin-Feng Xu
Xiao Zhou
Binhai Zhu
Michele A.A. Zito

Track (2) on Logic, Semantics, Specification, and Verification

Gilles Barthe
Marek Bednarczyk
Lars Birkedal
Andrzej Borzyszkowski
Ahmed Bouajjani
Colin Boyd
Julian Bradfield
Luca Cattani
Antonio Cerone
Witold Charatonik
Robert Colomb
Satoshi Fujita
Colin Fidge
Cedric Fournet
Jacques Garrigue
Robert Glück
Andrew D. Gordon
Therese Hardin
Daniel Hirschkoff
Paul Jackson
Michael Johnson
Jan Jurjens
Ed Kazmierczak
Yoshiki Kinoshita
Barbara Koenig
Beata Konikowska
Mark Lillibridge
Kamal Lodaya
Adam Malinowski
Bruno Marre
Richard Mayr
Toshiro Minami
Tetsuya Murai
Uwe Nestmann
Shin-ya Nishizaki
Michael Norrish
Christian Queinnec
R. Ramanujam
Peter J. Robinson
Christine Roeckl
Giuseppe Rosolini
Takafumi Sakurai
Philip Scott
Perdita Stevens

Colin Stirling
Paul Strooper
Eijiro Sumii
Wang-Chiew Tan
Moshe Vardi
Marek Zawadowski

Table of Contents

Keynote Plenary Talk 1

Reconciling Two Views of Cryptography
(The Computational Soundness of Formal Encryption) 3
Martín Abadi, Phillip Rogaway

Keynote Plenary Talk 2

Theory and Construction of Molecular Computers 23
Masami Hagiya

Keynote Plenary Talk 3

List Decoding: Algorithms and Applications 25
Madhu Sudan

Track (1) on Algorithms, Complexity and Models of Computation

Session 1.1

Approximation Algorithms for String Folding Problems 45
Giancarlo Mauri, Giulio Pavesi

An Index for Two Dimensional String Matching Allowing Rotations 59
Kimmo Fredriksson, Gonzalo Navarro, Esko Ukkonen

Session 1.2

Parallel Edge Coloring of a Tree on a Mesh Connected Computer 76
Chang-Sung Jeong, Sung-Up Cho, Sun-Chul Whang, Mi-Young Choi

Parallel Approximation Algorithms for Maximum Weighted Matching in
General Graphs 84
Ryuhei Uehara, Zhi-Zhong Chen

Invited Talk 1.1

It Is on the Boundary: Complexity Considerations for Polynomial Ideals . . 99
Ernst W. Mayr

Session 1.3

An Efficient Parallel Algorithm for Scheduling Interval Ordered Tasks 100
Yoojin Chung, Kunsoo Park, Hyuk-Chul Kwon

Task Distributions on Multiprocessor Systems 112
Evgeny V. Shchepin, Nodari N. Vakhania

Fast Interpolation Using Kohonen Self-Organizing Neural Networks 126
Olivier Sarzeaud, Yann Stéphan

Steganography Using Modern Arts
(Extended Abstract) 140
Carlo Blundo, Clemente Galdi

Session 1.4

Trade-Offs between Density and Robustness in Random Interconnection
Graphs 152
Philippe Flajolet, Kostas Hatzis, Sotiris Nikoletseas, Paul Spirakis

The $(\sigma + 1)$-Edge-Connectivity Augmentation Problem without Creating
Multiple Edges of a Graph 169
Satoshi Taoka, Toshimasa Watanabe

On the Hardness of Approximating Some NP-optimization Problems
Related to Minimum Linear Ordering Problem
(Extended Abstract) 186
Sounaka Mishra, Kripasindhu Sikdar

Maximum Clique and Minimum Clique Partition in Visibility Graphs 200
Stephan Eidenbenz, Christoph Stamm

Session 1.5

Real-Time Language Recognition by Alternating Cellular Automata 213
Thomas Buchholz, Andreas Klein, Martin Kutrib

Damage Spreading and μ-Sensitivity on Cellular Automata 226
Bruno Martin

Invited Talk 1.2

Discrepancy Theory and Its Application to Finance 243
Shu Tezuka

Session 1.6

Fully Consistent Extensions of Partially Defined Boolean Functions with
Missing Bits 257
Endre Boros, Toshihide Ibaraki, Kazuhisa Makino

Characterization of Optimal Key Set Protocols
(Extended Abstract) 273
Takaaki Mizuki, Hiroki Shizuya, Takao Nishizeki

On the Complexity of Integer Programming in the Blum-Shub-Smale Computational Model 286
Valentin E. Brimkov, Stefan S. Dantchev

On Logarithmic Simulated Annealing 301
Andreas Albrecht, Chak-Kuen Wong

Invited Talk 1.3

Hierarchical State Machines 315
Mihalis Yannakakis

Track (2) on Logic, Semantics, Sepecification, and Verification

Session 2.1

Ambient Groups and Mobility Types 333
Luca Cardelli, Giorgio Ghelli, Andrew D. Gordon

An Asynchronous, Distributed Implementation of Mobile Ambients 348
Cédric Fournet, Jean-Jacques Lévy, Alan Schmitt

Invited Talk 2.1

Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness 365
Naoki Kobayashi

Session 2.2

Local π-Calculus at Work: Mobile Objects as Mobile Processes 390
Massimo Merro, Josva Kleist, Uwe Nestmann

An Interpretation of Typed Concurrent Objects in the Blue Calculus 409
Silvano Dal Zilio

Session 2.3

A Higher-Order Specification of the π-Calculus 425
Joëlle Despeyroux

Open Ended Systems, Dynamic Bisimulation and Tile Logic 440
Roberto Bruni, Ugo Montanari, Vladimiro Sassone

Fibred Models of Processes: Discrete, Continuous, and Hybrid Systems (Extended Abstract) 457
Marcelo P. Fiore

On the Complexity of Bisimulation Problems for Pushdown Automata . . . 474
Richard Mayr

Session 2.4

A Type-Theoretic Study on Partial Continuations 489
Yukiyoshi Kameyama

Partially Typed Terms between Church-Style and Curry-Style 505
Ken-Etsu Fujita, Aleksy Schubert

Alternating Automata and Logics over Infinite Words
(Extended Abstract) 521
Christof Löding, Wolfgang Thomas

Hypothesis Support for Information Integration in Four-Valued Logics 536
Yann Loyer, Nicolas Spyratos, Daniel Stamate

Invited Talk 2.2

Masaccio: A Formal Model for Embedded Components 549
Thomas A. Henzinger

Session 2.5

A Single Complete Refinement Rule for Demonic Specifications 564
Karl Lermer, Paul Strooper

Reasoning about Composition Using Property Transformers and Their
Conjugates 580
Michel Charpentier, K. Mani Chandy

Invited Talk 2.3

Some New Directions in the Syntax and Semantics of Formal Languages . . 596
Gordon D. Plotkin

Panel Discussion on New Challenges for TCS

New Challenges for Theoretical Computer Science
(General Introduction to the Panel) 599
Jozef Gruska

Algorithm Design Challenges
(Position Statement) 602
Giorgio Ausiello

Quantumization of Theoretical Informatics
(Position Statement) 604
Jozef Gruska

Two Problems in Wide Area Network Programming
(Position Statement) 609
Ugo Montanari

New Challenges for Computational Models
(Position Statement) 612
Yoshihito Toyama

Towards a Computational Theory of Everything
(Position Statement) 614
Jiří Wiedermann

Open Lectures

On the Power of Interactive Computing 619
Jan van Leeuwen, Jiří Wiedermann

The Varieties of Programming Language Semantics
(Summary) 624
Peter D. Mosses

Author Index 629

Reconciling Two Views of Cryptography
(The Computational Soundness of Formal Encryption)

Martín Abadi[1] and Phillip Rogaway[2]

[1] Bell Labs Research, Lucent Technologies
abadi@lucent.com
www.pa.bell-labs.com/~abadi
[2] Department of Computer Science, University of California at Davis
rogaway@cs.ucdavis.edu
www.cs.ucdavis.edu/~rogaway

Abstract. Two distinct, rigorous views of cryptography have developed over the years, in two mostly separate communities. One of the views relies on a simple but effective formal approach; the other, on a detailed computational model that considers issues of complexity and probability. There is an uncomfortable and interesting gap between these two approaches to cryptography. This paper starts to bridge the gap, by providing a computational justification for a formal treatment of encryption.

1 Two Views of Cryptography

A fairly abstract view of cryptographic operations is often adequate for the design, analysis, and implementation of systems that use cryptography. For example, it is often convenient to ignore the details of an encryption function, and to work instead with a high-level description of what encryption is supposed to achieve.

At least two distinct abstract views of cryptographic operations have developed over the years. They are both consistent and they have both been useful, but they come from two mostly separate communities and they are quite different. In one of them, cryptographic operations are seen as functions on a space of symbolic (formal) expressions; their security properties are also modeled formally (e.g., [15,28,23,13,27,25,30,5,34,22,21,29]). In the other, cryptographic operations are seen as functions on strings of bits; their security properties are defined in terms of the probability and computational complexity of successful attacks (e.g., [18,11,37,19,17,16,8,9,7]).

There is an uncomfortable gap between these two views. In this paper, we call attention to this gap and start to bridge it. Representing the two views, we give two accounts of symmetric (shared-key) encryption: a simple one, based on a formal system, and a more elaborate one, based on a computational model. Our main theorem is a soundness result that relates the two accounts. It establishes that secrecy properties that can be proved in the formal world are true in the computational world. Thus, we obtain a computational justification for the formal treatment of encryption.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 3–22, 2000.
© Springer-Verlag Berlin Heidelberg 2000

As we relate the two accounts of encryption, we identify and make explicit some important choices. In particular, our main theorem excludes certain encryption cycles (such as encrypting a key with itself); we argue that this restriction is reasonable and necessary, although formal approaches generally ignore it. We also consider, for example, whether two ciphertexts may manifest if they were produced using the same key.

We believe that this paper suggests a profitable line of further research. It will take a significant research effort to relate the views of the people who invent, implement, break, and use cryptography. Continuing this work, it would be worthwhile to consider other cryptographic operations (such as signatures and hash functions), and to treat complete security protocols (such as key-distribution protocols) in addition to basic algorithms.

Connections between the formal view and the computational view should ultimately benefit both:

- These connections should strengthen the foundations of formal cryptology, and help in elucidating implicit assumptions and gaps in formal methods. They should confirm or improve the relevance of formal proofs about a protocol to concrete instantiations of the protocol, making explicit requirements on the implementations of cryptographic operations.
- Methods for high-level reasoning seem necessary for computational cryptology as it treats increasingly complex systems. Formal approaches suggest such high-level reasoning principles, and even permit automated proofs. In addition, some formal approaches capture naive but powerful intuitions about cryptography; a link with those intuitions should increase the appeal and accessibility of computational cryptology.

The next section is a more detailed discussion of the two views of cryptography; it also mentions related work. The rest of the paper proceeds as follows.

In Section 3, we define a class of expressions and an equivalence relation on those expressions. The expressions represent data, of the sort used in messages in security protocols; the equivalence relation captures when two pieces of data "look the same" to an adversary, treating encryption as a formal operator. These definitions are simple and purely syntactic. In particular, they do not require any notion of probability or computational complexity. They are typical of the definitions given in formal treatments of cryptography, and directly inspired by some of them.

Then, in Section 4, we present a computational model with strings of bits, probabilities, and complexities. In this model, we define secure encryption in terms of computational indistinguishability; our definition is similar, but not identical, to those of semantic security [18,7].

Finally, in Section 5, we relate equivalence to computational indistinguishability. We associate a probability ensemble with each formal expression; our main theorem establishes that equivalent expressions induce computationally indistinguishable ensembles. For example, the two expressions that represent two pieces of data encrypted under a fresh key will be equivalent. This equivalence can be read as a secrecy property, namely that the ciphertexts do not reveal the data.

Our main theorem implies that the two expressions correspond to computationally indistinguishable ensembles.

2 Background and Related Work

This section explains the two views of cryptography, still informally. It points to a few examples of work informed by those two views; there are many more. It also describes some related research.

The formal view. There is a large body of literature that treats cryptographic operations as purely formal. There, for example, the expression $\{M\}_K$ may represent an encrypted message, with plaintext M and key K. All of $\{M\}_K$, M, and K are formal expressions, rather than sequences of bits. Various functions can be applied to such expressions, yielding other expressions. One of them is decryption, which produces M from $\{M\}_K$ and K. Crucially, there is no way to recover M or K from $\{M\}_K$ alone. Thus, the idealized security properties of encryption are modeled (rather than defined). They are built into the model of computation on expressions.

This body of literature starts with the work of Dolev and Yao [15], DeMillo, Lynch, and Merritt [14], Millen, Clark, and Freedman [28], Kemmerer [23], Burrows, Abadi, and Needham [13], and Meadows [27]. It includes many different agendas and approaches, with a variety of techniques from the fields of rewriting, modal logic, process algebra, and others. Over the years, it has been used in the design of protocols, it has helped develop confidence in some existing protocols, and it has enabled the discovery of many attacks. It has also led to the development of effective methods and tools for automated protocol analysis; Lowe's and Paulson's works are two recent examples of these advances [25,30].

This formal perspective is fairly easy to apply for the users of encryption, for example for protocol designers. It captures an important intuition: an encrypted message reveals its plaintext only to those that know the corresponding decryption key, and it reveals nothing to others. This assertion is a simple (and simplistic) all-or-nothing statement, which can be conveniently built into a formal method. In particular, it does not require any notion of probability or of computational complexity: there is no need to say that an adversary may obtain some data but only with low probability or after an expensive computation. (However, probability and computational complexity are compatible with formalism, as demonstrated by the work of Lincoln et al. [24].)

Those who employ the formal definitions often warn that a formal proof does not imply a guarantee of security. One of the reasons for this caveat is the gap between the representation of encryption in a formal model and its concrete implementation. At the very least, it is desirable to know what assumptions about encryption are necessary. Those assumptions have seldom been stated explicitly, and not in enough detail to permit systematic discussion and rigorous proofs. We aim to remedy this situation.

A somewhat similar situation arises from the use of the random-oracle model in cryptography [10]: proofs that assume random oracles do not automatically

yield guarantees when the oracles are instantiated. However, we do not know of any natural examples where this gap has manifested itself.

The computational view. Another school of cryptographic research is based on the framework of computational complexity theory. A typical member of that school would probably say that the formal perspective is naive and disconnected from the realities of concrete cryptographic algorithms and protocols. Keys, plaintexts, and ciphertexts are all just strings of bits. An encryption function is just an algorithm. An adversary is essentially a Turing machine. Good protocols are those in which adversaries cannot do "something bad" too often and efficiently enough. These definitions are all about success probabilities and computational cost.

This computational view originates in the work of Blum and Micali [11], Yao [37], and Goldwasser and Micali [18]. It has strengthened the scientific foundations of cryptography, with a sophisticated body of definitions and theorems. It has also played a significant role in the development and study of particular protocols.

As an important example of the computational approach, we sketch a notion of secure encryption. Specifically, we choose to treat symmetric encryption, following Bellare, Desai, Jokipii, and Rogaway [7]. An encryption scheme is defined as a triple of algorithms $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Algorithm $\mathcal{K}$ (the key generator) makes random choices and then outputs a string k. Algorithm $\mathcal{E}$ (the encryption algorithm) flips random coins r to map strings k and m into a string $\mathcal{E}_k(m, r)$. Algorithm $\mathcal{D}$ (the decryption algorithm) maps strings k and c into a string $\mathcal{D}_k(c)$. We expect that $\mathcal{D}_k(\mathcal{E}_k(m, r)) = m$ for appropriate k, m, and r.

An adversary for an encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a Turing machine which has access to an oracle. We imagine realizing this oracle in one of two ways. In the first, the oracle chooses (once and for all) a random key k, and then encrypts each query x using $\mathcal{E}_k$ and fresh random coins. In the second, the oracle chooses (once and for all) a key k, and then, when presented with a query x, encrypts a string of 0 bits of equal length, using fresh random coins. An adversary's *advantage* is the probablity that the adversary outputs 1 when the oracle is realized in the first way minus the probablity that the adversary outputs 1 when the oracle is realized in the second way. An encryption scheme is regarded as good if an adversary's maximal advantage is a slow-growing function of the adversary's computational resources. This definition of security can be worked out rigorously and elegantly in both asymptotic and concrete versions (see Section 4.3). In any case, it is based on notions of probability and computational power.

Related work. The desire to relate the two views of cryptography is not entirely new (e.g., [3,20,26]). Nevertheless, there have been hardly any research efforts in this general direction. The work of Pfitzmann, Schunter, and Waidner [31] (which is simultaneous to ours and independent) starts from motivations similar to our own. It proves that some reactive, cryptographic systems satisfy

high-level (non-cryptographic) specifications, under computational assumptions on cryptographic operations. These results do not concern a formal model of cryptography, such as the one studied in this paper, but the relation to a formal model of cryptography is mentioned as an interesting subject for further work. Also relevant is the work of Lincoln, Mitchell, Mitchell, and Scedrov [24], which develops a rich process-algebraic framework that draws on both views of cryptography. Further afield, Abadi, Fournet, and Gonthier [1,2] and Lynch [26] relate the formal view of cryptography with higher-level (non-cryptographic) descriptions of security mechanisms. Finally, Volpano and Smith [35] analyze the complexity of attacking programs written in a simple, typed language; however, this language does not include cryptographic primitives.

As we compare two accounts of encryption, we arrive at the concept of which-key concealing encryption, with which ciphertexts do not manifest whether they were produced using the same key (see Section 4.2). Indepedently and concurrently, the work of Bellare, Boldyreva, Desai, and Pointcheval studies this concept from a different perspective [6].

3 Formal Encryption and Expression Equivalence

In this section we present the formal view of cryptography, specifically treating symmetric encryption. We describe the space of expressions on which encryption operates, and what it means for two expressions to be equivalent.

As explained in the introduction, the expressions represent data, of the sort used in messages in security protocols. Expressions are built up from bits and keys by pairing and encryption. The equivalence relation captures when two pieces of data "look the same" to an adversary that has no prior knowledge of the keys used in the data. For example, an adversary (with no prior knowledge) cannot obtain the key K from the ciphertexts $\{0\}_K$ and $\{1\}_K$; therefore, the adversary cannot decrypt and distinguish these ciphertexts, so they are equivalent. Similarly, the pairs $(0, \{0\}_K)$ and $(0, \{1\}_K)$ are equivalent. On the other hand, the pairs $(K, \{0\}_K)$ and $(K, \{1\}_K)$ are not equivalent, since an adversary can obtain K from them, then decrypt $\{0\}_K$ or $\{1\}_K$ and obtain 0 or 1, respectively, thus distinguishing the pairs. In this section, we formalize these informal arguments about equivalence; the soundness theorem of Section 5 provides a further justification for them.

3.1 Expressions

We write **Bool** for the set of bits $\{0, 1\}$. These bits can be used to spell out numbers and principal names, for example. We write **Keys** for a fixed, nonempty set of symbols disjoint from **Bool**. The symbols $K, K', K'', \ldots$ and $K_1, K_2, \ldots$ are all in **Keys**. Informally, elements of the set **Keys** represent cryptographic keys, generated randomly by a principal that is constructing an expression. Formally,

however, keys are atomic symbols, not strings of bits. We write **Exp** for the set of *expressions* defined by the grammar:[1]

$M, N ::=$	expressions
K	key (for $K \in \mathbf{Keys}$)
i	bit (for $i \in \mathbf{Bool}$)
(M, N)	pair
$\{M\}_K$	encryption (for $K \in \mathbf{Keys}$)

Informally, (M, N) represents the pairing of M and N, which might be implemented by concatenation plus markers, and $\{M\}_K$ represents the encryption of M under K, which might be implemented using a symmetric algorithm like DES, in CBC mode and with a random initialization vector. Pairing and encryption can be nested, as in the expression $(\{\{(0, K')\}_K\}_{K'}, K)$.

We emphasize that the elements of **Exp** are formal expressions (essentially, parse trees, abstract syntax trees) rather than actual keys, bits, concatenations, or encryptions. In particular, they are unambiguous: for example, (M, N) equals (M', N') if and only if M equals M' and N equals N', and it never equals $\{M'\}_K$. Similarly, $\{M\}_K$ equals $\{M'\}_{K'}$ if and only if M equals M' and K equals K'. However, according to definitions given below, $\{M\}_K$ and $\{M'\}_{K'}$ may be equivalent even when M and M' are different and when K and K' are different.

There are several possible extensions of the set of expressions:

- We could allow expressions of the form $\{M\}_N$, where an arbitrary expression N is used as encryption key.
- We could distinguish encryption keys from decryption keys, as in public-key cryptosystems.

These extensions are useful in modeling realistic protocols, but would complicate our definitions and theorems. We therefore leave them for further work.

It is also important to consider a restriction to the set of expressions. We say that K encrypts K' in M if there exists an expression N such that $\{N\}_K$ is a subexpression of M and K' occurs in N. For each M, this defines a binary relation on keys (the "encrypts" relation). We say that M is *cyclic* (or *acyclic*) if its associated relation is cyclic (or acyclic, respectively). For example, $\{K\}_K$ and $(\{K\}_{K'}, \{K'\}_K)$ are both cyclic, while $(\{K\}_{K'}, \{0\}_K)$ is acyclic.

Cycles, such as encrypting a key under itself, are a source of errors in practice (e.g., [36]); they also lead to weaknesses in common computational models,

[1] An equivalent way to define **Exp** is as the language generated by the context-free grammar with start symbol **E**, nonterminals **E** and **K**, terminals "0", "1", "(", ")", ",", "{", "}", and the set of elements in **Keys**, and the productions:

$$\mathbf{E} \longrightarrow 0 \mid 1 \mid (\mathbf{E}, \mathbf{E}) \mid \mathbf{K} \mid \{\mathbf{E}\}_{\mathbf{K}}$$
$$\mathbf{K} \longrightarrow K \quad \text{for each } K \in \mathbf{Keys}$$

as explained in Section 4. Moreover, cycles can often be avoided in practice—and they should generally be avoided given what is, and is not, known about them. The soundness theorem of Section 5 deals only with acyclic expressions. In contrast, cycles are typically permitted (without discussion) in formal methods.

3.2 Equivalence

Next we give a formal definition of equivalence of expressions. It draws on definitions from the works of Syverson and van Oorschot, Schneider, Paulson, and others [33,32,30]. Some of the auxiliary definitions concern how expressions can be analyzed and synthesized; such definitions are quite common in formal methods for protocol analysis. Equivalence relations are useful in semantics of modal logics: in such semantics, one says that two states in a computation "look the same" to a principal only if the principal has equivalent expressions in those states. Equivalence relations also appear in bisimulation proof techniques [4,12], where one requires that bisimilar processes produce equivalent messages.

First, we define an *entailment* relation $M \vdash N$, where M and N are expressions. Intuitively, $M \vdash N$ means that N can be computed from M. Formally, we define the relation inductively, as the least relation with the following properties:

- $M \vdash 0$ and $M \vdash 1$,
- $M \vdash M$,
- if $M \vdash N_1$ and $M \vdash N_2$ then $M \vdash (N_1, N_2)$,
- if $M \vdash (N_1, N_2)$ then $M \vdash N_1$ and $M \vdash N_2$,
- if $M \vdash N$ and $M \vdash K$ then $M \vdash \{N\}_K$,
- if $M \vdash \{N\}_K$ and $M \vdash K$ then $M \vdash N$.

This definition of $M \vdash N$ models what an attacker can obtain from M without any prior knowledge of the keys used in M. For example, we have

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash K_3$$

and

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash \{K_1\}_{K_2}$$

but not

$$(\{\{K_1\}_{K_2}\}_{K_3}, K_3) \vdash K_1 \qquad \text{(false)}$$

It is simple to derive a more general definition from this one: obtaining N from M with prior knowledge of K is equivalent to obtaining N from (M, K) with no prior knowledge.

Next, we introduce the box symbol $\Box$, which represents a ciphertext that an attacker cannot decrypt. We define the set **Pat** of *patterns* as an extension of the set of expressions, with the grammar:

$P, Q ::=$	patterns
K	key (for $K \in$ **Keys**)
i	bit (for $i \in$ **Bool**)

(P, Q)	pair
$\{P\}_K$	encryption (for $K \in \mathbf{Keys}$)
$\Box$	undecryptable

Intuitively, a pattern is an expression that may have some parts that an attacker cannot decrypt.

We define a function that, given a set of keys T and an expression M, reduces M to a pattern. Intuitively, this is the pattern that an attacker can see in M if the attacker has the keys in T.

$$\begin{aligned} p(K, T) &= K \qquad (\text{for } K \in \mathbf{Keys}) \\ p(i, T) &= i \qquad (\text{for } i \in \mathbf{Bool}) \\ p((M, N), T) &= (p(M, T), p(N, T)) \\ p(\{M\}_K, T) &= \begin{cases} \{p(M, T)\}_K & \text{if } K \in T \\ \Box & \text{otherwise} \end{cases} \end{aligned}$$

Further, we define a pattern for an expression without an auxiliary set T, but using the set of keys obtained from the expression itself.

$$\mathit{pattern}(M) \;=\; p(M, \{K \in \mathbf{Keys} \mid M \vdash K\})$$

Intuitively, this is the pattern that an attacker can see in M using the set of keys obtained from M. (As above, we assume that the attacker has no prior knowledge of the keys used in M, without loss of generality.) For example, we have

$$\mathit{pattern}((\{\{K_1\}_{K_2}\}_{K_3}, K_3)) \;=\; (\{\Box\}_{K_3}, K_3)$$

Finally, we say that two expressions are *equivalent* if they yield the same pattern:

$$M \equiv N \;\text{ if and only if }\; \mathit{pattern}(M) = \mathit{pattern}(N)$$

For example, we have:

$$\{\{K_1\}_{K_2}\}_{K_3}, K_3) \equiv (\{\{0\}_{K_1}\}_{K_3}, K_3)$$

since both expressions yield the pattern $(\{\Box\}_{K_3}, K_3)$.

We may view keys as bound names, subject to renaming (as in the spi calculus [5]). For example, although $(\{0\}_K, K)$ and $(\{0\}_{K'}, K')$ are not equivalent, we may say that they are equivalent up to renaming. More generally, we define *equivalence up to renaming*, $\cong$, as follows:

$$\begin{aligned} M \cong N \;\text{ if and only if }\; & \text{there exists a bijection } \sigma \text{ on } \mathbf{Keys} \\ & \text{such that } M \equiv N\sigma \end{aligned}$$

where $N\sigma$ is the result of applying σ as a substitution to N. Although this relation $\cong$ is looser than $\equiv$, our soundness theorem treats it smoothly, without difficulty. Therefore, we focus on $\cong$. In informal discussions, we often do not distinguish the two relations, calling them both equivalence.

3.3 Some Examples and Some Subtleties

In this section we give a few more examples. Some of the examples indicate assumptions and choices built into the definition of equivalence. These are fairly subtle but important, and it is useful to be explicit about them. We revisit them in Section 4.

- $0 \cong 0$, of course.
- $0 \not\cong 1$, of course.
- $\{0\}_K \cong \{1\}_K$.
- $(K, \{0\}_K) \not\cong (K, \{1\}_K)$, but $(K, \{(\{0\}_{K'}, 0)\}_K) \cong (K, \{(\{1\}_{K'}, 0)\}_K)$.
- $K \not\equiv K'$ and $K \cong K'$, since keys are subject to renaming with $\cong$ but not with $\equiv$.
- $\{0\}_K \cong \{1\}_{K'}$ and even $\{0\}_K \equiv \{1\}_{K'}$, although the two ciphertexts are under different keys.
- Similarly, $(\{K'\}_K, \{0\}_K) \cong (\{K'\}_K, \{1\}_{K'})$ and $(\{K'\}_K, \{0\}_K) \equiv (\{K'\}_K, \{1\}_{K'})$.
- $\{0\}_K \cong \{K\}_K$, despite the encryption cycle in $\{K\}_K$.
- $\{(((1,1),(1,1)),((1,1),(1,1)))\}_K \cong \{0\}_K$.
  Informally, we are assuming that a plaintext of any size can be encrypted, and that the size of the plaintext cannot be deduced from the resulting ciphertext without knowledge of the corresponding decryption key. This property justifies equivalences such as the one above, where the two plaintexts are of different sizes. In an implementation, it can be guaranteed by padding plaintexts up to a maximum size, and truncating larger expressions or mapping them to some fixed string (see Section 4).
  We could easily refine the equivalence relation to make it sensitive to sizes, for example by introducing a symbol $\Box_n$ for each size n. The resulting definitions would be heavier.
- $(\{0\}_K, \{0\}_K) \cong (\{0\}_K, \{1\}_K)$.
  Informally, we are assuming that an attacker who does not have a key cannot even detect whether two plaintexts encrypted under the key are identical. For example, the attacker should not be able to tell that the same plaintext appears twice under K in $(\{0\}_K, \{0\}_K)$, hence $(\{0\}_K, \{0\}_K) \cong (\{0\}_K, \{1\}_K)$. In an implementation, this sort of equivalence can be guaranteed by randomization of the encryption function (see Section 4).
  We could easily refine the equivalence relation to make it sensitive to message identities (for example as in [4]); but, again, the resulting definitions would be heavier.
- $(\{0\}_K, \{1\}_K) \cong (\{0\}_K, \{1\}_{K'})$.
  Informally, we are assuming that an attacker who does not have a key cannot even detect whether two ciphertexts use that same key. For example, the attacker should not be able to tell that the same key is used twice in $(\{0\}_K, \{1\}_K)$, hence $(\{0\}_K, \{1\}_K) \cong (\{0\}_K, \{1\}_{K'})$.
  Again, an alternative definition would be possible, with some complications.

4 The Computational View: Encryption Schemes and Indistinguishability

In this section we provide a computational treatment for symmetric encryption. First we describe the functions that constitute a symmetric encryption scheme, and then we describe when an encryption scheme should be called secure. Actually, there are a few different possibilities for defining security, and we discuss several of them. The notion that we focus on—which we call type-0 security—is stronger than the customary notion of security (that is, semantic security, and notions equivalent to it [18,7]). Nonetheless, one can achieve type-0 security under standard complexity-theoretic assumptions. We focus on type-0 security because it matches up with the formal definitions of Section 3. Other computational notions of security can be paired with analogous formal ones.

4.1 Preliminaries

Elements of an encryption scheme. Let $\mathsf{String} = \{0,1\}^*$ be the set of all finite strings, and let $|x|$ be the length of string x. Let Plaintext, Ciphertext, and Key be nonempty sets of finite strings. Let $\mathbf{0}$ be a particular string in Plaintext. Encrypting a string not in Plaintext will result in a ciphertext that decrypts to $\mathbf{0}$. We assume that if $x \in \mathsf{Plaintext}$ then $x' \in \mathsf{Plaintext}$ for all x' of the same length as x. Let Key be endowed with some fixed distribution. (If Key is finite, the distribution on Key is the uniform one.) Let Coins be a synonym for $\{0,1\}^\omega$ (the set of infinite strings), and Parameter (the set of *security parameters*) be a synonym for 1^* (the set of finite strings of 1 bits).

An *encryption scheme*, Π, is a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where

$$\begin{aligned}\mathcal{K} &: \mathsf{Parameter} \times \mathsf{Coins} \to \mathsf{Key}\\ \mathcal{E} &: \mathsf{Key} \times \mathsf{String} \times \mathsf{Coins} \to \mathsf{Ciphertext}\\ \mathcal{D} &: \mathsf{Key} \times \mathsf{String} \to \mathsf{Plaintext}\end{aligned}$$

and each algorithm is computable in time polynomial in the size of its input (but without consideration for the size of Coins input). Algorithm $\mathcal{K}$ is called the *key-generation* algorithm, $\mathcal{E}$ is called the *encryption* algorithm, and $\mathcal{D}$ is called the *decryption* algorithm. We usually write the first argument to $\mathcal{E}$ or $\mathcal{D}$, the key, as a subscript. When we omit mention of the final argument to $\mathcal{K}$ or $\mathcal{E}$ this indicates the corresponding probability space, or, when used as a set, the support of that probability space (that is, the strings which are output with nonzero probability). We require that for all $\eta \in \mathsf{Parameter}$, $k \in \mathcal{K}(\eta)$, and $r \in \mathsf{Coins}$, if $m \in \mathsf{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m,r)) = m$, while if $m \notin \mathsf{Plaintext}$ then $\mathcal{D}_k(\mathcal{E}_k(m,r)) = \mathbf{0}$. For example, the encryption function could treat an out-of-domain message as though it was $\mathbf{0}$. We insist that $|\mathcal{E}_k(x)|$ depends only on η and $|x|$ when $k \in \mathcal{K}(\eta)$.

The definition above is for probabilistic, stateless encryption. One can be a bit more general, allowing the encryption algorithm to maintain state. We do not pursue this generalization here.

Other basic concepts. A function $\epsilon : \mathsf{N} \to \mathsf{R}$ is *negligible* if $\epsilon(\eta) \in \eta^{-\omega(1)}$. This means that for all $c > 0$ there exists N_c such that $\epsilon(\eta) \leq \eta^{-c}$ for all $\eta \geq N_c$. An *ensemble* (or *probability ensemble*) is a collection of distributions on strings, $D = \{D_\eta\}$, one for each η. We write $x \stackrel{R}{\leftarrow} D_\eta$ to indicate that x is sampled from D_η. Let $D = \{D_\eta\}$ and $D' = \{D'_\eta\}$ be ensembles. We say that D and D' are *indistinguishable* (or *computationally indistinguishable*), and write $D \approx D'$, if for every probabilistic polynomial-time adversary A, the function

$$\epsilon(\eta) \stackrel{\text{def}}{=} \Pr[x \stackrel{R}{\leftarrow} D_\eta : \ A(\eta, x) = 1] - \Pr[x \stackrel{R}{\leftarrow} D'_\eta : \ A(\eta, x) = 1]$$

is negligible.

4.2 Aspects of Encryption-Scheme Security

In this section we consider some possible attributes of encryption schemes, and also consider encryption cycles. These issues already appear in Section 3 in a formal setting; here we explore them further in a computational setting.

Attributes (present or absent) of a secure encryption scheme. We single out three characteristics of an encryption scheme. The first and third are well-known, while the second seems not to have received attention till now.

- Repetition concealing vs. repetition revealing:
  Given ciphertexts c and c', can one tell if their underlying plaintexts are equal? If so, we call the scheme repetition revealing; otherwise, it is repetition concealing. A repetition-concealing scheme must be probabilistic (or stateful); making encryption schemes repetition concealing is one motivation for probabilistic encryption [18].
- Which-key concealing vs. which-key revealing:
  If one encrypts messages under various keys, can one tell which messages were encrypted under the same keys? If so, we call the scheme which-key revealing; otherwise, it is which-key concealing. Though standard instantiations of encryption schemes are which-key concealing, standard definitions for encryption-scheme security (like those in [18,7]) do not guarantee this. Demanding that an encryption scheme be which-key concealing is useful in contexts beyond that of the present paper (for example, in achieving forms of anonymity). The current work of Bellare et al. undertakes a thorough treatment of which-key concealing encryption [6].
- Message-length concealing vs. message-length revealing:
  Does a ciphertext reveal the length of its underlying plaintext? If so, we call the scheme message-length revealing; otherwise, it is message-length concealing. Most encryption schemes are message-length revealing. The reason is that implementing message-length concealing encryption invariably entails padding messages to some maximal length, and it may therefore be quite inefficient. Message-length concealing encryption is possible when the message space is finite, or when all ciphertexts are infinite streams (rather than finite strings as stated in our definitions).

These three characteristics are orthogonal, and all eight combinations make sense. Let us call these eight notions of security type-0, type-1, ..., type-7, with the numbering determined as follows: concealing corresponds to a 0 bit and revealing to a 1 bit, and we interpret the three characteristics above as a 3-bit binary number, the most significant bit being for repetition concealing or revealing, then which-key concealing or revealing, finally message-length concealing or revealing. With this terminology, the conventional concept of encryption-scheme security, ever since the work of Goldwasser and Micali [18], has been type-3 security: a ciphertext may reveal the length of the message and which key is being used, but it should not reveal if two ciphertexts are encryptions of the same message. However, this concept of security is not the only reasonable one.

Encryption cycles. Given a type-n ($n \in \{0, \ldots, 7\}$) secure encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, one can construct a type-n secure encryption scheme $\Pi' = (\mathcal{K}, \mathcal{E}', \mathcal{D}')$ with the following property: Π' would be completely insecure if the adversary were given (for example, as an additional input) even a single encryption $c \stackrel{R}{\leftarrow} \mathcal{E}'_k(k)$ of the underlying key k. Goldwasser and Micali were aware of this (in the public-key setting) when they published their work [18].

It is not only encrypting k under k that is problematic; longer cycles may also cause problems. For example, even if an encryption scheme is type-3 secure, it may not be safe to encrypt a message b under a key a and then, reversing the roles of a and b, to encrypt a under b. For all we know, the concatenation of the two ciphertexts might trivially reveal both a and b. For probabilistic encryption, for cycles of length greater than one, we do not have any example to demonstrate that this problem can actually arise, but the hybrid arguments [18,37] often used to prove encryption schemes secure, and which we use here, do not work in the presence of such cycles.

Therefore, as discussed in Section 3, we focus on expressions without encryption cycles. In return, we can rely on standard-looking definitions and tools in the computational setting.

4.3 Definitions of Encryption-Scheme Security (Types 0, 1, 3)

The formal treatment in Section 3 corresponds to type-0 security (repetition concealing, which-key concealing, and message-length concealing), so let us define this notion more precisely. An explanation of the notation follows the definition.

Definition 1 (Type-0 security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme, let $\eta \in$ Parameter be a security parameter, and let A be an adversary. Define*

$$\mathsf{Adv}^0_{\Pi[\eta]}(A) \stackrel{\text{def}}{=} \Pr\left[k, k' \stackrel{R}{\leftarrow} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(\cdot),\, \mathcal{E}_{k'}(\cdot)}(\eta) = 1\right] - \Pr\left[k \stackrel{R}{\leftarrow} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(\mathbf{0}),\, \mathcal{E}_k(\mathbf{0})}(\eta) = 1\right]$$

Encryption scheme Π is type-0 secure *if for every probabilistic polynomial-time adversary A, $\mathsf{Adv}^0_{\Pi[\eta]}(A)$ is negligible (as a function of η).*

We are looking at the difference of two probabilities.

- First, let us focus on the first probability. The quantity in brackets describes an experiment that is performed, and then an event. In this experiment, one first chooses two keys, k and k', independently, by running the key-generation algorithm $\mathcal{K}$. Then one runs adversary A, with two oracles: a left oracle f and a right oracle g. If the adversary asks the left oracle f a query $m \in \mathsf{String}$, the oracle returns a random encryption of m under key k. That is, the oracle computes $c \xleftarrow{R} \mathcal{E}_k(m)$ and returns c. If the adversary asks the right oracle g a query $m \in \mathsf{String}$, the oracle returns a random encryption of m under key k', similarly. Independent coins are used each time a string is encrypted (but the keys k and k' stay fixed).
- Next, let us consider the second probability. In this experiment, a single key k is selected by running the key-generation algorithm $\mathcal{K}$. The adversary again has two oracles, a left oracle f and a right oracle g, and these oracles again expect queries $m \in \mathsf{String}$. But now the oracles behave in the same way. When asked a query m, the oracles ignore the query, sample $c \xleftarrow{R} \mathcal{E}_k(\mathbf{0})$, and return c. Independent coins are used each time a string is encrypted (but the key k stays fixed).

The *type-0 advantage* is the difference in the above probabilities. One can imagine that the adversary is trying to distinguish a good encryption box from a false one. A good encryption box encrypts the specified query using the selected key. A false encryption box ignores the query and encrypts a fixed message under a fixed random key. Intuitively, a scheme is type-0 secure if no reasonable adversary can do a good job at telling apart the two encryption boxes on the basis of their input/output behavior.

Various other equivalent formalizations for type-0 encryption are possible. For example, it adds no power for there to be more than two oracles. (In the first experiment, each oracle would encrypt queries under its own key; in the second, every oracle would encrypt $\mathbf{0}$ under a common key.) Likewise, it takes away no power if $\mathcal{E}_{k'}(\cdot)$ is replaced with $\mathcal{E}_{k'}(\mathbf{0})$ in the first experiment.

We also give detailed definitions of type-1 and type-3 security; they resemble that of type-0 security. In these definitions, $\mathcal{E}_k(\cdot)$ is an oracle that returns $c \xleftarrow{R} \mathcal{E}_k(m)$ on input m, as above, and $\mathcal{E}_k(0^{|\cdot|})$ is an oracle that returns $c \xleftarrow{R} \mathcal{E}_k(0^{|m|})$ on input m.

Definition 2 (Type-1 security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme, let $\eta \in \mathsf{Parameter}$ be a security parameter, and let A be an adversary. Define*

$$\mathsf{Adv}^1_{\Pi[\eta]}(A) \stackrel{\text{def}}{=} \Pr\left[k, k' \xleftarrow{R} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(\cdot), \mathcal{E}_{k'}(\cdot)}(\eta) = 1\right] - \Pr\left[k \xleftarrow{R} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(0^{|\cdot|}), \mathcal{E}_k(0^{|\cdot|})}(\eta) = 1\right]$$

Encryption scheme Π is type-1 secure *if for every probabilistic polynomial-time adversary A, $\mathsf{Adv}^1_{\Pi[\eta]}(A)$ is negligible (as a function of η).*

Definition 3 (Type-3 security). *Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme, let $\eta \in \mathsf{Parameter}$ be a security parameter, and let A be an adversary. Define*

$$\mathsf{Adv}^3_{\Pi[\eta]}(A) \stackrel{\text{def}}{=} \Pr\left[k \xleftarrow{R} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(\cdot)}(\eta) = 1\right] - \Pr\left[k \xleftarrow{R} \mathcal{K}(\eta) :\ A^{\mathcal{E}_k(0^{|\cdot|})}(\eta) = 1\right]$$

Encryption scheme Π is type-3 secure *if for every probabilistic polynomial-time adversary A, $\mathsf{Adv}^3_{\Pi[\eta]}(A)$ is negligible (as a function of η).*

4.4 Achieving Type-0 and Type-1 Security With Standard Tools

Since type-3 security is standard but type-0 and type-1 security are not, we show that type-0 and type-1 security can be achieved using standard assumptions and constructions. Although this fact is not necessary for our soundness theorem, it provides support for the hypotheses of the theorem.

Block ciphers. Let $\beta \geq 1$ be a number (the blocksize) and let $\mathsf{Block} = \{0,1\}^\beta$. Let Key be a finite nonempty set. Then a *block cipher* is a function $E : \mathsf{Key} \times \mathsf{Block} \to \mathsf{Block}$ such that, for every $k \in \mathsf{Key}$, we have that $E_k(\cdot) = E(k, \cdot)$ is a permutation. Example block ciphers are DES and the emerging AES (Advanced Encryption Standard).

One measure of security for a block cipher is:

$$\mathsf{Adv}^{\text{prp}}_E(A) = \Pr\left[k \xleftarrow{R} \mathsf{Key} :\ A^{E_k(\cdot)} = 1\right] - \Pr\left[\pi \xleftarrow{R} \text{Perm}(\beta) :\ A^{\pi(\cdot)} = 1\right]$$

Here $\text{Perm}(\beta)$ denotes the set of all permutations on $\{0,1\}^\beta$. Informally, the adversary A is trying to distinguish the block cipher E, as it behaves on a random key k, from a random permutation π. We think of E as a good block cipher if $\mathsf{Adv}^{\text{prp}}_E(A)$ is small as long as A is of reasonable computational complexity.

Block cipher modes of operation. Block ciphers are the most common building block for making symmetric encryption schemes. Two well-known ways to do this are CBC mode and CTR mode. In CBC mode (with a random initialization vector), the encryption of a plaintext $x = x_1 \ldots x_n$ using key $k \in \mathsf{Key}$, where $n \geq 1$ and $|x_i| = \{0,1\}^\beta$, is $y_0 y_1 \ldots y_n$ where $y_0 \xleftarrow{R} \mathsf{Block}$ and $y_i = E_k(y_{i-1} \oplus x_i)$ for all $1 \leq i \leq n$. In CTR mode, the encryption of a plaintext x using key k is the concatenation of $r \xleftarrow{R} \mathsf{Block}$ with the xor of x and the $|x|$-bit prefix of the concatenation of $E_k(r)$, $E_k(r+1)$, $E_k(r+2)$, ... Here $r+i$ is the β-bit string that encodes the sum of r (treated as an unsigned number) and i, modulo 2^β. In [7], Bellare et al. establish the (type-3) security of these two modes of operation. Their results are quantitative, measuring how well one can attack the block cipher E in terms of how well one can attack the given encryption schemes based on E (in the sense of type-3 security).

CBC and CTR modes are which-key concealing. Even though the results just mentioned do not indicate that CBC mode or CTR mode are which-key concealing, these schemes are in fact which-key concealing and those results can be used to show it, as we now sketch. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme, let A be an adversary, and define

$$\mathsf{Adv}^{\mathrm{rand}}_{\Pi[\eta]} = \Pr\left[k \xleftarrow{R} \mathsf{Key}(\eta) :\ A^{\mathcal{E}_k(\cdot)}\ (\eta) = 1\right] - \Pr\left[k \xleftarrow{R} \mathsf{Key}(\eta) :\ A^{\$^{|\mathcal{E}_k(\cdot)|}}\ (\eta) = 1\right]$$

By $\$^{|\mathcal{E}_k(\cdot)|}$ we denote an oracle which, on input m, computes $c \xleftarrow{R} \mathcal{E}_k(m)$ and returns a random string of length $|c|$. (By an assumption stated above, $|c|$ depends only on η and $|m|$.) Informally, the adversary cannot tell if it is given a real encryption oracle or an oracle that returns a random string (of the appropriate length) in response to every query.

The proofs of security in [7] actually establish that CBC mode and CTR mode are good schemes according to $\mathsf{Adv}^{\mathrm{rand}}$, assuming that the underlying block cipher E is secure according to $\mathsf{Adv}^{\mathrm{prp}}$. To complete the picture, we claim that any good scheme according to $\mathsf{Adv}^{\mathrm{rand}}$ is also type-1 secure. (This claim is not hard to prove, though we omit doing so here.) Therefore, CBC mode and CTR mode (as defined above) are type-1 secure: repetition concealing, which-key concealing, but message-length revealing.

Hiding message lengths for type-0 security. Finally, we have to conceal message lengths. This step is standard, provided the message space is finite. Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a type-1 secure encryption scheme with $\mathsf{Plaintext} = \{0,1\}^*$. Let $\mathsf{Plaintext}' \subseteq \mathsf{String}$ be a finite set, with a particular element $\mathbf{0}'$. To make a type-0 secure encryption scheme we just encode all messages of $\mathsf{Plaintext}'$ into strings of some fixed length, and then encrypt these using $\mathcal{E}$. That is, we choose any convenient function $encode(\cdot)$ which (reversibly) takes strings in $\mathsf{Plaintext}'$ to a subset of $\{0,1\}^\ell$, for some number ℓ. The encryption scheme $\Pi' = (\mathcal{K}, \mathcal{E}', \mathcal{D}')$, with message space $\mathsf{Plaintext}'$, is defined by letting $\mathcal{E}'_k(m) = \mathcal{E}_k(encode(m))$ for $m \in \mathsf{Plaintext}'$, setting $\mathcal{E}'_k(m) = \mathcal{E}'_k(\mathbf{0}')$ for $m \notin \mathsf{Plaintext}'$, and defining $\mathcal{D}'$ in the obvious way. Type-1 security of Π immediately implies type-0 security of Π'.

5 The Computational Soundness of Formal Equivalence

In this section we relate the two views of cryptography. We proceed in two steps. First, we show how to associate an ensemble to an expression M, given an encryption scheme Π. Then we show that, under appropriate assumptions, equivalent expressions give rise to indistinguishable ensembles.

5.1 Associating an Ensemble to an Expression

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme and let $\eta \in \mathsf{Parameter}$ be a security parameter. We associate to each formal expression $M \in \mathbf{Exp}$ a distribution

```
algorithm INITIALIZE_η(M)
  for K ∈ Keys(M) do τ(K) ←R K(η)

algorithm CONVERT(M)
  if M = K where K ∈ Keys then
    return ⟨ τ(K), "key" ⟩
  if M = b where b ∈ Bool then
    return ⟨ b, "bool" ⟩
  if M = (M1, M2) then
    return ⟨ CONVERT(M1), CONVERT(M2), "pair" ⟩
  if M = {M1}K then
    x ←R CONVERT(M1)
    y ←R E_τ(K)(x)
    return ⟨ y, "ciphertext" ⟩
```

Fig. 1. *How to map (probabilistically) an expression M to a string* $\textsc{Convert}(M)$*, given an encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ and a security parameter η.*

on strings $[\![M]\!]_{\Pi[\eta]}$, and thereby an ensemble $[\![M]\!]_{\Pi}$. This association constitutes a concrete semantics for expressions (in the style of programming-language semantics or logic semantics); it works as follows:

- First, we map each key symbol K that occurs in M to a string of bits $\tau(K)$, using the key generator $\mathcal{K}(\eta)$.
- We map the formal bits 0 and 1 to standard string representations for them.
- We obtain the image of a formal pair (M, N) by concatenating the images of the components M and N.
- We obtain the image of a formal encryption $\{M\}_K$ by calculating $\mathcal{E}_{\tau(K)}(x)$, where x is the image of M.
- In all cases, we tag string representations with their types (that is, "key", "bool", "pair", "ciphertext") in order to avoid any ambiguities.

This association is defined more precisely in Figure 1. In the figure, we write $\mathit{Keys}(M)$ for the set of all key symbols that occur in M, and write $\langle x_1, \ldots, x_k \rangle$ for an ordinary string encoding of $x_1, \ldots, x_k$. The auxiliary initialization procedure $\textsc{Initialize}_\eta(M)$ maps every key symbol in $\mathit{Keys}(M)$ to a unique key $\tau(K)$. The probability of a string in $[\![M]\!]_{\Pi[\eta]}$ is that induced by the algorithm $\textsc{Convert}(M)$ of Figure 1.

5.2 Equivalence Implies Indistinguishability

Our theorem is that equivalent expressions correspond to indistinguishable ensembles, assuming that the expressions are acyclic and that the underlying encryption scheme is type-0 secure.

Theorem 1. *Let M and N be acyclic expressions and let Π be a type-0 secure encryption scheme. Suppose that $M \cong N$. Then $[\![M]\!]_{\Pi} \approx [\![N]\!]_{\Pi}$.*

The proof of this theorem is a hybrid argument, as in [18,11,37]. One must be particularly careful in forming the hybrids, relying on acyclicity. Because of the generality of the claim, the description of the hybrid argument is somewhat complex and long. Therefore, we omit the proof in the present version of this paper. We only give a few simple examples instantiating the claim that $M \cong N$ implies $[\![M]\!]_\Pi \approx [\![N]\!]_\Pi$:

- Since $0 \cong 0$, we conclude that $[\![0]\!]_\Pi \approx [\![0]\!]_\Pi$. The two ensembles being compared put all the probablity mass on a single point, $\langle\, 0, \text{“bool”}\,\rangle$.
- Since $K \cong K'$, we conclude that $[\![K]\!]_\Pi \approx [\![K']\!]_\Pi$. The two ensembles being compared are identical: they are induced by the key generator $\mathcal{K}$.
- Since $\{0\}_K \cong \{1\}_K$, we conclude that $[\![\{0\}_K]\!]_\Pi \approx [\![\{1\}_K]\!]_\Pi$. This indistinguishability is nontrivial: it relies on the assumption that the encryption scheme is type-0 secure.
- Although $\{0\}_K \cong \{K\}_K$, we cannot conclude anything about how $[\![\{0\}_K]\!]_\Pi$ may relate to $[\![\{K\}_K]\!]_\Pi$, because of the encryption cycle in $\{K\}_K$.

Reconsidering some of the other examples of Section 3.3 can also be instructive.

One may wonder whether a converse to this theorem holds, that is, whether indistinguishability implies equivalence. This converse fails, for fairly trivial reasons: if $\langle\, 0, \text{“bool”}\,\rangle, \langle\, 1, \text{“bool”}\,\rangle \notin \mathsf{Plaintext}$, then the same ensemble is associated with the expressions $(K, \{0\}_K)$ and $(K, \{1\}_K)$, but these expressions are not equivalent. We have not explored in detail whether the converse holds when $\mathsf{Plaintext}$ is large enough.

6 Conclusions

The formal approach to cryptography often deals with simple, all-or-nothing assertions about security. The computational approach, on the other hand, makes a delicate use of probability and computational complexity. However, one may intuit that the formal assertions are valid in computational models, if not absolutely at least with high probability and against adversaries of limited computational power. In this paper, we develop this intuition, applying it to the study of encryption. We prove that the intuition is correct under substantial but reasonable hypotheses. This study of encryption is a step—perhaps modest but hopefully suggestive—toward treating security protocols and complete systems, and toward combining the sophistication of computational models with the simplicity and power of formal reasoning.

Acknowledgments

Jan Jürjens suggested improvements to the presentation of a draft of this paper. Phillip Rogaway was supported in part by NSF CAREER Award CCR-9624560, and by MICRO award 98-129.

References

1. Martín Abadi. Protection in programming-language translations. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 868–883. Springer-Verlag, July 1998. Also Digital Equipment Corporation Systems Research Center report No. 154, April 1998.
2. Martín Abadi, Cédric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. In *Proceedings of the Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 105–116, June 1998.
3. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47, 1997.
4. Martín Abadi and Andrew D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, Winter 1998.
5. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, January 1999. An extended version appeared as Digital Equipment Corporation Systems Research Center report No. 149, January 1998.
6. Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Anonymous encryption. Unpublished manuscript, 2000.
7. Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: analysis of the DES modes of operation. In *Proceedings of 38th Annual Symposium on Foundations of Computer Science (FOCS 97)*, 1997.
8. Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer-Verlag, 1994. To appear in *Journal of Computer and System Sciences*.
9. Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—CRYPTO '94*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 1993.
10. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
11. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo random bits. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 82)*, pages 112–117, 1982.
12. Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Proof techniques for cryptographic processes. In *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science*, pages 157–166, July 1999.
13. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, February 1989.
14. Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic protocols. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, 1982.
15. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, March 1983.

16. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, 1987.
17. Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or All languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(3):691–729, 1991.
18. Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, April 1984.
19. Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attack. *SIAM Journal on Computing*, 17:281–308, 1988.
20. James W. Gray, III, Kin Fai Epsilon Ip, and King-Shan Lui. Provable security for cryptographic protocols—exact analysis and engineering applications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 45–58, 1997.
21. James W. Gray, III and John McLean. Using temporal logic to specify and verify cryptographic protocols (progress report). In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 108–116, 1995.
22. R. Kemmerer, C. Meadows, and J. Millen. Three system for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, Spring 1994.
23. Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, May 1989.
24. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
25. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.
26. Nancy Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 14–29, 1999.
27. Catherine Meadows. A system for the specification and analysis of key management protocols. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 182–195, 1991.
28. Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.
29. John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 141–151, 1997.
30. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.
31. Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Cryptographic security of reactive systems (extended abstract). *Electronic Notes in Theoretical Computer Science*, 32, April 2000.
32. Steve Schneider. Security properties and CSP. In *IEEE Symposium on Security and Privacy*, pages 174–187, 1996.

33. Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocol logics. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28, 1994.
34. F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings 1998 IEEE Symposium on Security and Privacy*, pages 160–171, May 1998.
35. Dennis Volpano and Geoffrey Smith. Verifying secrets and relative secrecy. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 268–276, 2000.
36. David Wagner. Re: Security of DES key encrypted with its self???? On the Web at http://www.cs.berkeley.edu/~daw/my-posts/key-as-iv-broken-again, 1996.
37. Andrew C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS 82)*, pages 80–91, 1982.

Theory and Construction of Molecular Computers

Masami Hagiya

Graduate School of Science, University of Tokyo,
hagiya@is.s.u-tokyo.ac.jp,
Project home page: http://hagi.is.s.u-tokyo.ac.jp/MCP/

Molecular computing is a research area which aims to explore the potential of computation by molecules. Although the idea dates back to Feynman, it became realistic only when Adleman succeeded in solving a Hamiltonian path problem using DNA. The research community for the investigation and development of *DNA Based Computers* was then quickly formed by groups in the United States, Europe, Japan, etc.

The Japanese Molecular Computer Project, *Theory and Construction of Molecular Computers,* began in 1996 and will end in 2001. This project has produced a large number of experimental results, which verify the feasibility of basic operations for molecular computations and will help design a large scale molecular computer in the near future. In this, the last year of the project, a medium-scale DNA computer in which most reactions are executed by robot hands is under construction. A large number of theoretical results have also been produced by the project. Such theoretical studies either followed previous experimental results or initiated further experimental investigations.

I strongly believe that molecular computing should aim to explore the computational power inherent in molecular reactions. It should not be restricted to solving combinatorial problems by means of massive parallelism. In particular, a deep understanding of the computational power of molecules can be applied to many areas of information technology, biotechnology, and nanotechnology.

In this talk, after summarizing the achivements of our molecular computer project, I will suggest a set of perspectives to place the theory of molecular computing. This set can be roughly classified as follows.

- Studies which seek to propose or develop computational models suitable for describing molecular reactions. These include recent membrane models, which incorporate cell structure, in addition to simple molecular reactions.
- Studies on the computability of such computational models. In particular, achieving universal computability has been the major research interest in these studies. These include already classic ideas to build Turing machines using DNA.
- Studies on the complexity of such computational models. The trade-off between the number of computational steps and the amount of molecules necessary is a typical research issue. Note that molecular reactions should always be analyzed as parallel computations.

In addition to the above main stream of research, there are a large number of studies for analyzing the overall fidelity and efficiency of molecular computations.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 23–24, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Related studies on encoding performance and design are also very active and important for reducing error and enhancing efficiency. Although the design of good encodings can be formulated as a combinatorial problem, it has recently been recognized that a thermodynamical treatment is essential for analyzing the error mechanisms inherent in molecular reactions.

According to statistical thermodynamics, the behavior of molecular reactions, and therefore that of molecular computations, is inherently probabilistic, even if the possibility of error is ignored. I believe that it is appropriate to analyze molecular computations as probabilistic processes, taking into account the physical properties of molecular reactions. Studies on the complexity of molecular computations should therefore include probabilistic analyses based on statistical thermodynamics. Such physical analyses will deepen our understanding of molecular reactions and will be applied to many areas as fundamental knowledge.

List Decoding: Algorithms and Applications*

Madhu Sudan**

Department of Electrical Engineering and Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139.
madhu@mit.edu

Abstract. Over the years coding theory and complexity theory have benefited from a number of mutually enriching connections. This article focuses on a new connection that has emerged between the two topics in the recent years. This connection is centered around the notion of "list-decoding" for error-correcting codes. In this survey we describe the list-decoding problem, the algorithms that have been developed, and a diverse collection of applications within complexity theory.

1 Introduction

The areas of coding theory and complexity theory have had a long and sustained history of interesting connections. Early work on computation in the presence of noise built on these connections. Recent successes of complexity theory, showing IP=PSPACE and giving PCP characterizations of NP have relied on connections with coding theory either implicitly or explicitly. The survey article of Feigenbaum [10] gives a detailed account of many connections and consequences.

Over the last few years a new strain of connections has emerged between coding theory and complexity theory. These connections are different from the previous ones in that they rely especially on the qualitative strength of the decoding algorithms; and in particular on the ability to recover from large amounts of noise. The first work in this vein seems to be that of Goldreich and Levin [12], whose work describes (implicitly) an error-correcting code and gives a highly efficient algorithm to decode the code from even the slightest non-trivial amount of information. They use the algorithm to give a generic construction of hard-core predicates from an arbitrary one-way function. Subsequently, there have been a number of other such results providing even more powerful decoding algorithms and deriving other applications from these algorithms to complexity theory.

The main theme common to these works is the application of a new notion for decoding of error-correcting codes called *list decoding*. List decoding formalizes the notion of error-correction, when the number of errors is potentially very large.

* This survey is a fuller version of a previous one by the author that appeared in *SIGACT NEWS*, Volume 31, Number 1, pp. 16-27, March 2000.

** Supported in part by a Sloan Foundation Fellowship and NSF Career Award CCR-9875511.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 25–41, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Borrowing the terminology from the area of information communication, recall that to transmit information over a noisy channel, the transmitter transmits a codeword of an error-correcting code. This transmitted word is corrupted by the noisy channel, and the receiver gets some corrupted word that we will call "the received word." If the number of errors that occur during transmission is very large, then the received word may actually be closer to some codeword other than the transmitted one. Under the mandate of list-decoding, the receiver is required to compile a list of all codewords within a reasonable sized Hamming ball around the received word (and not just *the* nearest one). The list-decoding is declared to be successful if this list includes the transmitted word.

This notion of list decoding was proposed by Elias [9] in the 1950's. However till recently no non-trivial[1] list decoding algorithms were known for any error-correcting code. Of late, we have seen a spurt of efficient list-decoding algorithms; and equally interestingly, a diverse collection of applications of these list-decoders to complexity theoretic problems. In this survey, we describe some of these results. First we start with some definitions.

2 Error-Correcting Codes and List-Decoding

A block error-correcting code $\mathcal{C}$ is a collection of strings called codewords, all of which have the same length, over some finite alphabet Σ. The three basic parameters describing the code are the size of the alphabet, denoted q; the length of the codewords n; and an information parameter k, where the number of codewords is q^k. Such a code is succinctly referred to as an $(n, k)_q$ code[2]. If Σ has a field structure imposed on it, then Σ^n may be viewed as a vector space. If additionally $\mathcal{C}$ forms a linear subspace of Σ^n, then $\mathcal{C}$ is termed a linear code and denoted an $[n, k]_q$ code[3]. Almost all codes dealt with in this article will be linear codes.

In order to ensure that the code helps in the recovery from errors, one designs codes in which any two codewords differ from each other in large number of locations. Formally, let the *Hamming distance* between strings x and y from Σ^n, denoted $\Delta(x, y)$, be the number of coordinates where x and y differ from each other. The the distance of a code $\mathcal{C}$, typically denoted $d(\mathcal{C})$, is the minimum, over all pairs of non-identical codewords in $\mathcal{C}$, of the distance between the pair.

One of the first observations that can be made about a code $\mathcal{C}$ with distance d is that it can unambiguously correct $\frac{d-1}{2}$ errors, i.e., given any word $r \in \Sigma^n$, there exists at most one codeword $c \in \mathcal{C}$ such that $\Delta(r, c) \leq \frac{d-1}{2}$. It is also easy to find a word r such there exist two codewords at distance $\frac{d+1}{2}$ from it, so one can not improve the error bound for unambiguous decoding. However it was realized

[1] Here triviality is used to rule out both brute-force search algorithms and the unique decoding algorithms.

[2] Sometimes in the literature this would be referred to as an $(n, q^k)_q$ code, where the second parameter counts the number of messages as opposed to say the "length" of the message.

[3] Note the subtle change in the notation.

early on that unambiguous decoding is not the only useful notion of recovery from error. Elias [9] proposed the notion of *list decoding* in which a decoding algorithm is expected to output a list of all codewords within a given distance e from a received word $r \in \Sigma^n$. If the list of words output is relatively small, then one could consider this to be a reasonable recovery from error. Algorithmically, this problem is stated as follows:

Definition 1 (List decoding problem for a code $\mathcal{C}$).
INPUT: *Received word $r \in \Sigma^n$, error bound e.*
OUTPUT: *A list of all codewords $c_1, \ldots, c_m \in \mathcal{C}$ that differ from r in at most e places.*

As usual, the goal is to solve the list decoding problem efficiently: i.e., in time polynomial in n. However this is only possible if the output size is polynomially bounded in n. This motivates the following, purely combinatorial, question.

Definition 2 (List decoding problem: Combinatorial version).
For every c, determine the function $e_c(n, k, d, q)$ such that for every $(n, k)_q$ code $\mathcal{C}$ of distance $d(\mathcal{C}) = d$, and for every received word $r \in \Sigma^n$, there are at most $(qn)^c$ codewords in the Hamming ball of radius e around r.

We would like to study the asymptotic growth of e_c when we say fix the ratio k/n and d/n and let $n \to \infty$. If it makes sense, we would then like to study e_∞, the limit of e_c as $c \to \infty$. It turns out that e_∞ is fairly well-understood and this will be described in Section 3. Somewhat coincidentally, for a variety of codes, the list decoding problem can be solved in polynomial time provided $e < (1 - o(1)) \cdot e_\infty$. These results will be described in Section 4.

Before concluding this section, we present one more version of the algorithmic list-decoding problem that has been studied in the literature. This version is motivated by the question: Are there sub-linear time list-decoding algorithms for any error-correcting code? At first glance, linear time in n appears to be a lower bound on the running time since that is the amount of time it takes to read the input, or even the time to output one codeword. However, by specifying the input implicitly and allowing the output also to be specified implicitly, one is no longer subject to these trivial lower bounds on computation time. The notion of implicit representation of the input can be formalized by using an "oracle" to represent the input—when queried with an index i, the oracle responds with the ith bit of the received word. The notion of implicit representation of the output is somewhat more involved. Roughly we would like each element of the output list to be described succinctly by an efficient program that computes any one coordinate of the codeword. However these programs are allowed to be randomized; furthermore, they are allowed to make oracle calls to the implicit input when attempting to compute any one coordinate of the output. The notion of implicit representation of an output (codeword) is thus formalized by the concept of "probabilistic oracle machines," machines that are allowed to make oracle calls (to the received word). Under this formalism, the list decoding problem may now be rephrased as:

Definition 3 (List decoding problem: Implicit version).
IMPLICIT INPUT: *Oracle access to received word* $r : \{1 \ldots, n\} \to \Sigma$, *error bound* e.
OUTPUT: *A list of all codewords* $c_1, \ldots, c_m \in \mathcal{C}$, *represented implicitly by probabilistic oracle machines* $M_1, \ldots, M_m$ *working with oracle access to* r, *that differ from* r *in at most* e *places. For every* $i \in \{1, \ldots, m\}$ *and* $j \in \{1, \ldots, n\}$, M_i *satisfies the property that* $\Pr[M_i^{(r)}(j) = c_i(j)] \geq \frac{2}{3}$.

We remark that these implicit representations have now become common and useful in the theory of computation (e.g., in works on program testing/self-correcting, PCPs etc.). They allow for more modular use of algorithmic ideas; and results expressed in these terms deserve attention. It turns out that for the list decoding problem highly efficient solutions exist in this model for some codes—essentially in time polynomial in $\log n$. This efficiency translates into some very useful applications in complexity, and this will be described in the forthcoming sections.

3 Status of the Combinatorial Problem

We first sketch the status of the combinatorial problem described above. It is easily seen that $e_\infty(n, k, d, q) \geq e_0(n, k, d, q) = \frac{d-1}{2}$ (the unambiguous error-correction radius). Also, if the parameters are such that an $(n, k)_q$ code with distance d does exist, then it is also possible to get one along with a received word that has exponential in d codewords at distance d from it. Informally, this suggests $e_\infty(n, k, d, q) \leq d$ (though to be formal, we should first let n go to infinity, and then let c go to infinity!). Thus it seems reasonable to believe that e_c may be of the form αd, where α is some universal constant between $1/2$ and 1, and possibly a function of c. Unfortunately, the answer is not so simple: e_c turns out to be a function also of n and q and surprisingly is not very dependent on c. Roughly, (if q is very large), then $e_c \approx n - \sqrt{n(n-d)}$. (Even the task of performing a sanity check on this expression, i.e., to verify that $d/2 \leq n - \sqrt{n(n-d)} \leq d$, takes a few moments!) Some insight into this expression: If $d = o(n)$, then $n - \sqrt{n(n-d)}$ is well approximated by $d/2$. However, if d is large, i.e., $d = n - o(n)$, then the bound on e is also $n - o(n)$ and so e is well-approximated by d. We conclude that in the former case, the list-decoding radius is limited by the "half the distance" barrier, while in the latter case, it is not so limited.

The following theorem essentially refines the above expression to take into account small values of q. Recall that the "Plotkin bound" of coding theory shows that error-correcting codes with $d \geq (1 - 1/q) \cdot n$ have only polynomially many codewords and hence are not very interesting. Thus it makes sense to compare d and e as fractions of n' rather than n. The theorem statement replaces all occurrences of n in the expression above by $n' = (1 - 1/q) \cdot n$.

Theorem 4.

1. *Let* n, k, d, q, e *satisfy* $d \leq n'$ *and* $e < \left(1 - \sqrt{1 - \frac{d}{n'}}\right) \cdot n'$ *where* $n' = \left(1 - \frac{1}{q}\right) \cdot n$. *Then, for every* $(n, k)_q$ *code* $\mathcal{C}$ *with* $d(\mathcal{C}) \geq d$ *and for every received word* r, *there are at most* qn^2 *codewords within a Hamming distance of* e *from* r.
2. *For every* n, d, q, e, ϵ *such that* $\epsilon > 0$, $d < n'$ *and* $e \geq (1+\epsilon) \cdot \left(1 - \sqrt{1 - \frac{d}{n'}}\right) \cdot n'$ *where* $n' = \left(1 - \frac{1}{q}\right) \cdot n$, *there exists a (non-linear)* $(n, k)_q$ *code* $\mathcal{C}$ *of distance at least* d *and a received word* r, *such that there are exponentially many codewords (with the exponent growing with* ϵn*) within a Hamming distance of* e *from* r.

Note: The theorem above appears explicitly in [13]. The crucial direction, Part (1) above, is a q-ary extension of the "Johnson bound" in coding theory. Johnson proves this bound only for the binary case, but the extension to the q-ary case seems to be implicitly known to the coding theory community [27, Chapter 4, page 301].

Proof [Sketch]:

1. (Following a proof of Guruswami and Sudan [18].) Fix the q-ary alphabet Σ, the received word $r \in \Sigma^n$ and let $c_1, \ldots, c_m$ be codewords within a Hamming distance of e from r. Let $\bar{e}$ denote the average distance (averaged over i) of c_i from r. Note $\bar{e} \leq e$. The main steps in the proof are (1) Associate with Σ, q orthonormal vectors in q-dimensional real space. Without loss of generality these maybe the coordinate vectors. (2) Use this association to embed the vectors r and $c_1, \ldots, c_m$ in $\mathcal{R}^{qn}$. (3) Pick i and j in $\{1, \ldots, m\}$ at random (with replacement) and consider the expectation of the inner product $\langle c_i - r, c_j - r \rangle$. Since c_i and c_j are close to r and further are not very close to each other, this inner product is small in expectation. Specifically the expected value is at most $2\bar{e} - d + \frac{d}{m}$. On the other hand the vectors $c_i - r$ are not small (they are non-zero in an average of $2\bar{e}$ locations) and have non-negative inner product in each coordinate. Some elementary manipulation (which involved studying the location of the non-zero coordinates, their signs and an application somewhere of the Cauchy-Schwartz inequality) shows that the expected inner product is at least $\frac{q\bar{e}^2}{(q-1)n}$. This yields the inequality

$$\frac{q\bar{e}^2}{(q-1)n} \leq 2\bar{e} - d + \frac{d}{m}$$

   which, in turn, yields the bound in Part (1) of the Theorem.
2. (Following Goldreich et al.[13]) Let r be the all zeroes vector. Pick $c_1, \ldots, c_m$ independently as follows: In each coordinate c_j is chosen randomly (and independently of all else) to be 0 with probability $1 - \frac{e}{n}$ and chosen to be a random non-zero element of Σ otherwise. The probability that there exists

a pair within distance d is easily bounded from above by $m^2\exp(-\epsilon n)$. Thus it is possible to pick exponentially many codewords that are mutually at distance at least d while with high probability are within distance e to the received vector.

∎

Theorem 4 yields an asymptotically tight result on e_∞. To study this asymptotic limit, let us minimize some of the parameters above. First notice that k does not play any role in Part (1) of the theorem. So let $e_c(n, \cdot, d, q)$ denote the minimum over k of $e_c(n, k, d, q)$. Now further fix $d = \delta \cdot (1 - \frac{1}{q}) \cdot n$ and let $q = q(n)$ be any function of n. Now let

$$\epsilon_c(\delta) = \lim_{n\to\infty} \frac{e(n, \cdot, \delta(1 - \frac{1}{q})n, q)}{(1 - \frac{1}{q})n}.$$

Let $\epsilon_\infty(\delta) = \lim_{c\to\infty} \epsilon_c$. Then Theorem 4 above can be summarized as:

Corollary 5. *For $\delta \in [0, 1]$, $\epsilon_2 = \epsilon_\infty(\delta) = 1 - \sqrt{1 - \delta}$.*

In the next section, we will describe algorithmic results which come close to matching the combinatorial results above.

4 Specific Codes and Performance of List-Decoding Algorithms

We start by introducing the reader to a list of commonly used (and some not so commonly used) error-correcting codes. In the first five codes below, q will be assumed to be a prime power, and Σ will be a finite field on q elements.

Hadamard codes. For any k, the Hadamard code $\mathcal{H}_k$ is a $(n = q^k, k)_q$ code with distance $(1 - \frac{1}{q})q^k$, obtained as follows: The message is a k dimensional vector over Σ, denoted α. The codeword is indexed by space of k-dimensional vectors. The β-th symbol in the encoding of α is their inner product, that is $\sum_{i=1}^{k} \alpha_i \cdot \beta_i$.

(Generalized) Reed Solomon codes. Here the message is thought of as specifying polynomial of degree at most $k-1$ by giving its k coefficients. The encoding evaluates the polynomial at n distinct points in the finite field. (It follows that q has to be at least n.) The fact that two distinct degree $k-1$ polynomials may agree on at most $k-1$ points yields that the distance is at least $n-k+1$. The fact that there do exist distinct degree $k-1$ polynomials that agree at any given subset of $k-1$ places shows that the distance is exactly $n-k+1$.

Reed Muller codes. Reed Muller codes may be viewed as a common generalization of Reed Solomon codes and Hadamard codes. For parameters m and $l < q$, the Reed Muller code has $k = \binom{m+l}{m}$ and $n = q^m$. The message is viewed as specifying a polynomial of total degree at most l over m variables. The encoding gives the evaluation of this polynomial at every possible input. For $l < q$, the codewords are at a distance of at least $(1 - l/q)n$ from each other.

Algebraic geometric codes. These codes are also generalizations of the generalized Reed Solomon codes. Description of the construction of these codes is out of scope. All we will say is that they yield $(n, k)_q$ codes with distance at least $d = n - k - n/(\sqrt{q} - 1)$ when q is a square. In contrast a random linear code of dimension k has distance approximately $n - k - n/\log q$. Thus the algebraic geometric codes asymptotically beat the distance achieved by the random code, provided q is large enough!

Concatenated codes. This term refers to any code obtained by a certain process, called concatenation of codes, that derives a new code from two given codes. Specifically, given an "outer" code over a q^k-ary alphabet and an "inner" code of dimension k over a q-ary alphabet, the concatenated codeword corresponding to a given message is obtained by first encoding the message using the outer code, and then encoding each symbol of the resulting string by the inner code. If the outer code is an $(n_1, k_1)_{q^{k_2}}$ code and the inner code is an $(n_2, k_2)_q$ code, then the concatenated code is an $(n_1 n_2, k_1 k_2)_q$ code. If d_1 is the distance of the outer code and d_2 is the distance of the inner code, then the concatenated code has minimum distance $d_1 d_2$. In this section we will consider codes obtained by concatenating a Reed-Solomon, Reed-Muller or Algebraic-Geometry code as the outer code with a Hadamard code as the inner code.

Chinese remainder codes. These codes are an aberration in the class of codes we consider in that they are not defined over any single alphabet. Rather the i-th symbol is from an alphabet of size p_i, where $p_1, \ldots, p_n$ are n distinct primes arranged in increasing order. The messages of this code are integers between 0 and $K - 1$, where $K = \prod_{j=1}^{k} p_j$. The encoding of a message is the n-tuple of its residues modulo $p_1, \ldots, p_n$. In the coding theory literature, this code is often referred to as the Redundant Residue Number System code. As an easy consequence of the Chinese Remainder Theorem, we have that the message can be inferred given any k of the n residues making this a code of distance $n-k+1$. If $p_1 \approx p_n \approx p$, then one may view this code as approximately an $(n, k)_p$ code.

4.1 List Decoding Results: Explicit Version

For some families of codes, it is possible to get algorithms that perform list-decoding in polynomial time for e, the number of errors, as large as the bound in Theorem 4. The following theorem lists this family of results.

Theorem 6. *Let $\mathcal{C}$ be an $(n, k)_q$ code with designed distance*[4] *$d = \delta n'$, where $n' = (1 - 1/q)n$. Further, if $\mathcal{C}$ is either a (1) Hadamard code, (2) Reed-Solomon code, (3) Algebraic-geometric code, (4) Reed-Solomon or algebraic-geometry code*

[4] In at least two cases, that of algebraic-geometry codes and algebraic-geometry codes concatenated with Hadamard code, the code designed to have distance d, may turn out to have larger minimum distance. Typical decoding algorithms are unable to exploit this extra bonus, and work only against the designed distance of the code; in fact, there may be no short proof of the fact that the code has this larger minimum distance. This explains the term "designed distance" of a code.

concatenated with a Hadamard code, or (5) Chinese remainder code, then it has a polynomial time list decoding algorithm that decodes from $e < (1 - \sqrt{1-\delta})n'$ *errors.*

Proofs of any of these results is out of scope. We will simply give some pointers here.

Remarks:

1. Note that the result for Hadamard codes is trivial, since this code has only n codewords. Thus a brute force search algorithm that lists all codewords and then evaluates their distance against the received word to prune this list, runs in time $O(n^2)$.
2. An algorithm for list-decoding the Reed-Solomon codes when $e < n - \sqrt{2n(n-d)}$ was given by Sudan[31] based on earlier work of Ar, Lipton, Rubinfeld, and Sudan [2]. For the case of explicit list-decoding problem this was the first non-trivial list-decoder that was constructed for any code. The tight result above is from the work of Guruswami and Sudan [17].
3. The first list-decoder for algebraic-geometry codes was given by Shokrollahi and Wasserman [29]. There error bound matched that of [31]. The tight result above is again from [17].
4. It is easy to combine non-trivial list-decoders for the outer code and inner code to get some non-trivial list decoding of a concatenated code. However, such results will not obtain the tight result described above. The tight result above is due to Guruswami and Sudan [18].
5. A list decoder correcting $n - \sqrt{2kn\frac{\log p_n}{\log p_1}}$ errors for the Chinese remainder codes was given by Goldreich, Ron, and Sudan [14]. Boneh [7] recently improved this bound to correct $n - \sqrt{kn\frac{\log p_n}{\log p_1}}$ errors. Even more recently Guruswami, Sahai, and Sudan [16] improve this to correct $n - \sqrt{nk}$ errors.

4.2 List Decoding Results: Implicit Version

For the implicit list-decoding problem, some fairly strong results are known for the cases of Hadamard codes, Reed-Muller codes and consequently for concatenated codes. We describe these results in the next two theorems. For the case of binary Hadamard codes, Goldreich and Levin [12] gave a list-decoding algorithm when the received word is specified implicitly. They consider the case when the number of errors is arbitrarily close to the limit of Theorem 4, and in particular takes the form $e = (\frac{1}{2} - \gamma)n$. They give a randomized algorithm whose running time is $\text{poly}(\log n, \frac{1}{\gamma})$ and reconstructs explicitly a list of all messages (note that message lengths are $O(\log n)$ for the Hadamard code) that come within this error of the received word. Subsequently, this algorithm was generalized to general q-ary Hadamard codes by Goldreich, Rubinfeld, and Sudan [13]. This yields the following theorem.

Theorem 7. *There exists a probabilistic list decoding algorithm in the implicit input model for Hadamard codes that behaves as follows: For an $(n,k)_q$ code $\mathcal{C}$, given oracle access to a received word r, the algorithm outputs a list that includes all messages that lie within a distance of e from the received word. The running time of the algorithm is a polynomial in $\log n, \log q$ and $\frac{n}{n-\frac{q}{q-1}e}$.*

For the case of Reed-Muller, equally strong list-decoding results are known, now with the output representation also being implicit. Arora and Sudan [3] provided such a list-decoder provided the error bound satisfies $e < n(1-(1-d/n)^{\epsilon})$, for some positive ϵ. Sudan, Trevisan, and Vadhan [32] improved this bound to a tighter bound of $e < (1-\sqrt{1-d/n})n$, thus yielding the following theorem.

Theorem 8. *There exists a probabilistic list decoding algorithm in the implicit input and implicit output model for Reed-Muller codes that behaves as follows: For a $(n = q^m, k = \binom{m+l}{l})_q$ Reed-Muller code $\mathcal{C}$, given oracle access to a received word r, the algorithm outputs a list of randomized oracle programs that includes one program for each codeword that lies within a distance of e from the received word, provided $e < (1 - O(\sqrt{l/q}))n$. The running time of the algorithm is a polynomial in m, l and $\log n$.*

As pointed out earlier. it is easy combine list-decoding algorithms for outer and inner codes to get a list-decoding algorithm for a concatenated code. By concatenating a Reed-Muller code with some appropriately chosen Hadamard code, one also obtains the following result, which turns out to be a handy result for many applications. In fact, all results of Section 5 use only the following theorem.

Theorem 9. *For every q, ϵ and k, if $n \geq \mathrm{poly}(k, q, \frac{1}{\epsilon})$ there exists an $(n,k)_q$ code with a polynomial time list-decoding algorithm for errors up to $(1-1/q-\epsilon)n$. Furthermore, the algorithm runs in time polynomial in $\log k$ and $1/\epsilon$ if the input and output are specified implicitly.*

The above result, specialized to $q = 2$ is described explicitly in [32]. The general codes and list-decoding algorithm can be inferred from their proof.

5 Applications in Complexity Theory

Algorithms for list-decoding have played a central role in a variety of results in complexity theory. Here we enumerate some (all?) of them.

Hardcore predicates from one-way permutations. A classical question lying at the very foundations of cryptography is the task of extracting one hard Boolean function (predicate), given any hard one-way function. Specifically given a function $f : \{0,1\}^k \to \{0,1\}^k$ that is easy to compute but hard to invert, obtain a predicate $P : \{0,1\}^k \to \{0,1\}$ such that $P(x)$ is hard to predict given $f(x)$. Blum and Micali [6] showed how it was possible to extract one such hard

predicate from the Discrete Log function and used it to construct pseudo-random generators. A natural question raised is whether this ability to extract hard predicates is special to the Discrete Log function, and if not, could such a hard predicate be extracted from every one-way function f.

At first glance this seems impossible. In fact, given any predicate P, it is possible to construct one-way functions f such that $f(x)$ immediately gives $P(x)$. However, this limitation is inherited from the deterministic nature of P. Goldreich and Levin [12] modify the setting to allow the predicate P to be randomized. Specifically, they allow the predicate P to be a function of x and an auxiliary random string r. P is considered hardcore for f if $P(x, r)$ is hard to predict with accuracy better than $\frac{1}{2} + \epsilon$ given $f(x)$ and r. (The function P is said to be predictable with accuracy α if the output of some polynomial sized circuit agrees P on α fraction of the inputs.) They show that this minor modification to the problem statement suffices to construct hardcore predicates from any one-way function.

One parameter of some interest in the construction of hardcore predicates is the length of the auxiliary random string. Let l denote this parameter. The initial construction of [12] (which was based on their list-decoding algorithm for the Hadamard code) sets $l = k$. Impagliazzo [19] gives an substantial improvement to this parameter, achieving $l = O(\log k + \log \frac{1}{\delta})$, by using the list-decoders for Reed-Solomon and Hadamard codes. It turns out that both constructions can be described as a special case of a generic construction using list-decodable codes. The construction goes as follows: Let $\mathcal{C}$ be a $(n, k)_2$ binary code as given by Theorem 9 with ϵ set to some poly(δ). Then the predicate $P(x, r) = (\mathcal{C}(x))_r$ (the rth bit of the encoding of x) is as hard as required. The proof follows modularly from the list-decodability property of $\mathcal{C}$. Specifically, if for some x, the prediction of the circuit agrees with $P(x, \cdot)$ for a $\frac{1}{2} + \epsilon$ fraction of the values of $i \in \{1, \ldots, n\}$, then one can use the list decoder to come up with a small list of candidates that includes x. Further, the knowledge of $f(x)$ tells us how to find which element of the list is x. The hardness of inverting f thus yields that there are not too many x's for which the circuit can predict $P(x, \cdot)$ with this high an accuracy. Now to see the effectiveness of Theorem 9, note that the extra input has length $\log n$, which by the theorem is only $O(\log k + \log \frac{1}{\delta})$.

Aside: Recall that the early results of Blum and Micali [6] and Alexi, Chor, Goldreich, and Schnorr [1] that gave hardcore predicates for specific one-way functions (namely, Discrete Log and RSA) actually use $l = 0$ extra randomness. It would be interesting to see if these specific results can also be explained in terms of list-decoding.

Predicting witnesses for NP-search problems. Consider an NP-complete relation such as 3-SAT. Kumar and Sivakumar [24], based on earlier work of Gal, Halevi, Lipton, and Petrank [11], raise the question of whether it is possible to efficiently construct a string x that has non-trivial proximity to a witness of the given instance. For general relations in NP they show that if some string with distance $\frac{1}{2} + \epsilon$ can be found; then NP=P. They show this result using the list-decoding algorithms for Reed Solomon and Hadamard codes. Again this

result can be explained easily using Theorem 9 as follows: Construct an NP relation whose instances are, say, instances of satisfiability but whose witnesses are encodings, using a code obtained from Theorem 9, of satisfying assignments. Given a string that has close proximity to a valid witness, one can recover a small set of strings one of which includes the witness.

Amplifying hardness of Boolean functions. One of the recent success stories in complexity theory is in the area of finding complexity theoretic assumptions that suffice to derandomize BPP. In a central result in this direction, Impagliazzo and Wigderson [22], show that a strong form of the assumption "E does not have subexponential sized circuits" implies BPP=P. One important question raised in this line of research is on amplification of the hardness of Boolean functions. Specifically, given a Boolean function $f : \{0,1\}^l \to \{0,1\}$, transform it into a Boolean function $f' : \{0,1\}^{l^{O(1)}} \to \{0,1\}$ such that if no small circuit computes f, then no small circuit computes f' on more than $\frac{1}{2} + \epsilon$ fraction of the inputs.

[22] give such a transformation which goes through a sequence of of transformations: one from Babai, Fortnow, Nisan and Wigderson [4], one from Impagliazzo [20], and a new one original to [22]. Again this step can be modularly achieved from error-correcting codes efficiently list-decodable under the implicit input/output model, as follows (from Sudan, Trevisan, and Vadhan [32]): Think of f as a 2^l bit string and encode this string using an error correcting code. Say the encoded string is a $2^{l'}$ bit string. Then this function can be thought of as the truth table of a function $f' : \{0,1\}^{l'} \to \{0,1\}$. It follows from the list-decodability properties of the error-correcting code that f' is highly unpredictable. Specifically, suppose C is a circuit predicting f'. Then C is an implicit representation of a received word that is close to f'; thus list-decoding, in the implicit output model, yields a small circuit computing f' (and with some work, a small circuit encoding f). The strength of this transformation is again in its efficiency. For example, Impagliazzo, Shaltiel, and Wigderson [21], note that this construction is also significantly more efficient in some parameters and use this aspect in other derandomizations of BPP.

Direct product of NP-complete languages. Let SAT be the characteristic function of the satisfiability language. I.e., $\mathrm{SAT}(\phi) = 1$ if ϕ is a satisfiable formula and 0 otherwise. Let SAT^l be the l-wise direct product of the SAT function. I.e., it takes as input l formulae $\phi_1, \ldots, \phi_l$ and outputs the l-bit vector $\mathrm{SAT}(\phi_1), \ldots, \mathrm{SAT}(\phi_l)$. Clearly SAT^l is at least as hard to compute as SAT. Presumably it is much harder. In fact if SAT were hard to compute on more than $1 - \delta$ fraction of the instances chosen from some distribution, then SAT^l would be hard to compute with probability more than $(1 - \delta)^l$ on the product distribution. Unfortunately, no NP-complete problem is known to be NP-hard when the inputs are chosen at random. In the face of this lack of knowledge, what can one say about SAT^l? This topic is studied in the complexity theory literature under the label of membership comparability. Sivakumar [30] gives a nice hardness for this problem. He shows that if it is even possible to efficiently compute the least amount of information about SAT^l, for $l(n) = O(\log n)$ then NP=RP. Specifically, if some polynomial time algorithm, on input an instance $\boldsymbol{\phi}$

of SAT^l, rules out even one string out 2^l as the value of $\mathrm{SAT}^l(\phi)$, then it can be used to decide satisfiability. Sivakumar [30] uses the list-decodability properties of the Reed Solomon codes and a version of Sauer's lemma. Simplifying the proof slightly it is possible to get it as a direct consequence of Theorem 9 applied to $q = 2^l$ and $\epsilon = 2^{-2l}$, without use of Sauer's Lemma (see [18].

Permanent of random matrices. In a striking result, Lipton [25], showed how it is possible to use the fact that the permanent is a low-degree polynomial to conclude the following. If it is easy to compute the permanent of an $n \times n$ matrix modulo a prime $p > n$, with high probability when the matrix is chosen at random, then it is also easy to compute the permanent of any $n \times n$ matrix modulo p. One of the first results to establish the average-case hardness of a computationally hard problem, this result laid down the seed for a series of interesting results in complexity theory including IP=PSPACE and the PCP characterizations of NP.

Subsequent results strengthened the average case hardness of the permanent to the point where it suffices to have an algorithm that computes the permanent modulo p on an inverse polynomially small fraction of the matrices, as shown by Cai, Pavan and Sivakumar [8]. Their result uses the list-decoding algorithm for Reed Solomon codes. Independently Goldreich, Ron and Sudan [13] strengthened this result in different direction. They show it suffices to have an algorithm that computes the permanent correctly with inverse polynomial probability when both the matrix and the prime are chosen at random. Their result uses the list decoding algorithm for the Reed Solomon codes as well as that for the Chinese Remainder code. It turns out that the techniques of [8] extend to this problem also, thus giving an alternate proof that does not use the list-decoder for the Chinese remainder code (but still uses the list-decoder for the Reed-Solomon code).

6 Sample of Algorithms

Here we attempt to briefly sketch the algorithmic ideas needed to get, say Theorem 9 (and thus covering all applications of Section 5). To get this result, we need the list-decoder for Reed-Solomon codes and Reed-Muller codes and some glue to patch the details. The reader is warned that this section is highly sketchy and many details are skimmed over without explicit notice.

6.1 Decoding Reed-Solomon Codes

Say we have an $(n, k)_q$ Reed-Solomon code obtained by taking degree $k-1$ polynomials and evaluating them at points $x_1, \ldots, x_n \in \Sigma$, where Σ is a field on q elements. Note that the list decoding problem here turns into the problem: Given n pairs $\{(x_1, r_1), \ldots, (x_n, r_n)\}$, find all degree $k-1$ polynomials p such that $p(x_i) = r_i$ for at least $n - e$ values of i.

We describe the algorithm for the case when $e < n - k\sqrt{n}$. The algorithm works in two steps. (1) Find a non-zero bivariate polynomial $Q(x, r)$ of degree

at most $\sqrt{n}$ in x and r, such that $Q(x_i, r_i) = 0$ for every i. (2) Factor Q into irreducible factors. For every irreducible factor of the form $r - q(x)$, check if $q(x_i) = r_i$ for at least $n - e$ value of i and output it if so.

First note that both steps can be implemented efficiently. The first step amounts to solving a homogeneous linear system to find a non-trivial solution, and hence can be solved efficiently. The second step is implementable efficiently as a consequence of efficient factorization algorithms for multivariate polynomials given by Chistov and Grigoriev, Kaltofen, or Lenstra (see survey article by Kaltofen [23] for pointers).

To verify correctness, we first need to verify that step (1) will return some polynomial Q. This is true since Q has more than n coefficients and thus the homogeneous linear system has more variables than constraints; and hence has a non-trivial solution. Now let p be a degree $k-1$ polynomial and $S \subseteq \{1, \ldots, n\}$ be a set of cardinality at least $n - e$ such that $p(x_i) = r_i$ for every $i \in S$. We claim $r - p(x)|Q(x, r)$. We prove this using the "division algorithm over unique factorization domains" which says this is the case iff $Q(x, p(x)) = 0$. To see this, let $g(x) = Q(x, p(x))$ and note that for every $i \in S$, $g(x_i) = Q(x_i, p(x_i)) = Q(x_i, r_i) = 0$. But g is a polynomial of degree at most $k\sqrt{n}$ that is zero on more than $k\sqrt{n}$ points. Hence g is identically zero, as required. Thus we have that $y - p(x)$ does divide Q and hence p will be included in the output list.

6.2 Implicit Decoding of Reed-Muller Codes

We first describe a solution for the case when the error is relatively small (small enough to guarantee unique solutions). Say we have access to the received word r as an oracle mapping Σ^m to Σ. Say p is a polynomial of degree l that agrees with r in $n-e$ places. We wish to design a (randomized) oracle program that computes p. Suppose we wish to compute $p(i)$ for $i \in \Sigma^m$. Pick $j \in \Sigma^m$ at random and consider the line $l = \{l(\theta) = (1-\theta)i + \theta j | \theta \in \Sigma\}$. We note that p restricted to this line, i.e., the function $p|_l(\theta) = p(l(\theta))$ is a univariate polynomial in θ of degree at most l. Further the value we are interested in $p|_l(0)$. The crucial observation here is that for any fixed $\theta \neq 0$, $l(\theta)$ is a random point in Σ^m and thus $r(l(\theta)$ is very likely to equal to $p|_l(\theta)$. Thus applying the univariate polynomial reconstruction algorithm (i.e. the list decoding algorithm for Reed-Solomon codes) to the points $\{(\theta, r(l(\theta)))|\theta \in \Sigma\}$ is very likely (over the random choice of j) to yield the polynomial $p|_l$; evaluating this at 0 yields $p(i)$. Summarizing, the randomized oracle program, call it $C^{(r)}$, that implicitly describes p works as follows: (1) Picks $j \in \Sigma^l$ at random and sets $l = \{(1-\theta)i + \theta j | \theta \in \Sigma\}$. (2) Uses the univariate reconstruction algorithm to compute (explicitly) the polynomial $p|_l(\cdot)$. (3) Outputs $p|_l(0)$.

In attempting to extend this algorithm to higher error, the main problem faced is the lack of information about p. Above, we used the fact that p had been implicitly specified by the oracle r and the fact that it was a degree l polynomial. But when e is large, many polynomials may agree with r on $n - e$ places, and the polynomial p is not yet specified uniquely. In other words, if $p_1, \ldots, p_t$ are all the polynomials that agree with r in $n-e$ places, then it is easy

to extend the above algorithm into one that outputs a small set that includes the values $\{p_1(i), \ldots, p_t(i)\}$. However it is hard to create the oracle for, say $p = p_1$. Among other things, it is not clear what distinguishes p_1 from any of the other polynomials in the list. To create this distinction, we need some additional information about p_1. Say we knew the value of p_1 at some point j, and let $p_1(j) = \sigma$. Then we may modify the algorithm of the previous paragraph to get a new one, call it $A^{(r)}_{j,\sigma}$, as follows: (1') Set $l = \{(1-\theta)i + \theta j\}$. (2') Use univariate reconstruction algorithm to compute a list of univariate polynomials $q_1, \ldots, q_t$ that agree with r on approximately $1 - e/n$ fraction of the points on the line l. (3') If there exists a polynomial q_k in this list such that $q_k(1) = \sigma$, then output $q_k(0)$, else do anything.

It can be shown that with high probability over the choice of j, $A^{(r)}_{j,p(j)}$ correctly computes p for most choices of i. This part requires some analysis and we will skip it (see [32]). Combining this with the self-corrector of the first paragraph, we get that $C^{(A^{(r)}_{j,p(j)})}$ is a probabilistic oracle machine for p. If a running time of poly(q) does not bother us, we may simply guess $p(j)$ by running through all possible choices; else better methods can give us a shorter list of candidates.

It may be easily verified that the running time of the decoder is polynomial in q and m (while n is q^m). For careful settings of q and m, the running time becomes polylogarithmic in n.

6.3 Decoding Concatenated Codes

Finally, it is easy to guess a simple strategy for list-decoding concatenated codes, given list-decoding algorithms for the outer and inner code. We describe a natural algorithm, without giving any proofs. However the proofs can be worked out as an exercise. The decoding algorithm for the concatenation of an $(n_1, k_1)_{q^{k_2}}$ outer code with an $(n_2, k_2)_q$ inner code may work as follows: Given a q-ary string of length $n_1 n_2$, first list-decode the n_1 strings of length n_2 corresponding to the inner code. In each case pick a random element of the list and thus create a q^{k_2}-ary string of length n_1 corresponding to the outer code. List decode this string. Repeat if necessary.

Now to see why these ideas suffice to yield Theorem 9, we take the code to be the concatenation of an outer Reed-Muller code and inner Hadamard code, with careful choice of code parameters. The list-decoder for the Reed-Muller code has already been described. For our purposes, the brute-force list-decoder for Hadamard codes will suffice at the inner level. By choosing appropriate thresholds for the list-decoding of the inner code, some relatively straightforward analysis yields Theorem 9.

7 Concluding Thoughts

By now we have seen many applications of algorithms for list-decoding. The notion of list-decoding itself, never mind the algorithmic results, is a very important one for complexity theory. The recent beautiful result of Trevisan [34],

gives strong evidence of the role that this theme can play in central questions in complexity/extremal combinatorics. (For those few of you who may have missed this development, Trevisan showed how to construct a strong family of extractors by combining binary codes that have very good combinatorial list-decoding properties, with a pseudo-random generator of Nisan and Wigderson [26]. This construction and its successors, see Raz, Reingold and Vadhan [28], and Impagliazzo, Shaltiel, and Wigderson [21], reach optimal characteristics for various choices of parameters.) We hope other applications of list-decoding will continue to emerge as the notion becomes more popular.

We conclude with some open questions relating to combinatorial list-decoding performance of error-correcting codes. The combinatorial question relating to list-decoding posed in this survey was chosen carefully to allow for a tight presentation of results (in Theorem 4 and its corollary). However, it is much more interesting to study list-decoding characteristics of specific codes and here we know very little (Part (1) of Theorem 4 applies, of course, but Part (2) is irrelevant to specific codes). For example, Ta-Shma and Zuckerman [33], have shown that the random (non-linear) code has polynomially many codes in any ball of radius e, for e very close to the minimum distance of the code. The existence of such codes with good list-decoding properties raises the question of whether such codes exist with small description size; and if so can they be constructed and/or decoded efficiently. One could ask such questions about the classes of codes described in this paper. For example, what is the largest error e for an $(n, k)_q$ Reed-Solomon code for which the Hamming ball of radius e around any received word has only poly(n) codewords. The best known bound is still given by Part (1) of Theorem 4. In fact the following question remains very interesting: Let $\epsilon_\infty^{\mathrm{Lin}}(\delta)$ be defined analogously to $\epsilon_\infty(\delta)$, however restricted to linear codes. As before we know $1 - \sqrt{1-\delta} \leq \epsilon_\infty^{\mathrm{Lin}}(\delta) \leq \delta$. However we know very little beyond this point. (In some recent work in progress, Guruswami, Håstad, Sudan, and Zuckerman [15], have shown that the analogous quantity $\epsilon_c^{\mathrm{Lin}}(\delta)$ is strictly smaller than δ for every choice of δ and c, however the difference in their proof vanishes as $c \to \infty$. Thus a number of questions relating to the combinatorics of the list-decoding problem remain open. Depending on the answers to these, a number of algorithmic challenges could also open up. Thus the area seems rife for further exploration.

Finally, a word of warning. This survey is very much a reflection of the author's current state of knowledge, or lack thereof. As the state of this knowledge improves, the survey will hopefully get updates and if so the updated copy will be available from the author's website `http://theory.lcs.mit.edu/~madhu`.

Acknowledgments

I'd like to thank Oded Goldreich, Venkatesan Guruswami, Johan Håstad, Luca Trevisan, Salil Vadhan, and David Zuckerman for sharing with me many of their thoughts and ideas on the topic of list-decoding.

References

1. Werner Alexi, Benny Chor, Oded Goldreich, and Claus P. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194-209, April 1988.
2. Sigal Ar, Richard J. Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from erroneous data. *SIAM Journal on Computing*, 28(2): 487–510, April 1999.
3. Sanjeev Arora and Madhu Sudan. Improved low degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4–6 May 1997.
4. László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.
5. Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In *7th Annual Symposium on Theoretical Aspects of Computer Science*, volume 415 of Lecture Notes in Computer Science, pages 37-48, Rouen, France, 22-24 February 1990. Springer.
6. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.
7. Dan Boneh. Finding smooth integers in short intervals using CRT decoding. (To appear) *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 21-23 May 2000.
8. Jin-Yi Cai, A. Pavan, and D. Sivakumar. On the hardness of the permanent. In *16th International Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Trier, Germany, March 4–6 1999. Springer-Verlag.
9. Peter Elias. List decoding for noisy channels. In *1957-IRE WESCON Convention Record*, Pt. 2, pages 94-104, 1957.
10. Joan Feigenbaum. The use of coding theory in computational complexity. In *Proceedings of Symposia in Applied Mathematics*, R. Calderbank (ed.), American Mathematics Society, Providence, pages 203–229, 1995.
11. Anna Gal, Shai Halevi, Richard Lipton, and Erez Petrank. Computing from partial solutions In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, Atlanta, Georgia, 4-6 May 1999.
12. Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15–17 May 1989.
13. Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries—the highly noisy case. Technical Report TR98-060, Electronic Colloquium on Computational Complexity, 1998. Preliminary version in FOCS '95.
14. Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, pages 225–234, Atlanta, Georgia, 1-4 May 1999.
15. Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Untitled Manuscript, January 2000.
16. Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. Untitled Manuscript, January 2000.
17. Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6): 1757–1767, September 1999.

18. Venkatesan Guruswami and Madhu Sudan, Low-rate codes with high error correction capabilities. (To appear) *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland, Oregon, 21-23 May 2000.
19. Russell Impagliazzo. Personal Communication, July 1997.
20. Russell Impagliazzo. Hard-core distributions for somewhat hard problems. In *36th Annual Symposium on Foundations of Computer Science*, pages 538–545, Milwaukee, Wisconsin, 23–25 October 1995. IEEE.
21. Russell Impagliazzo, Ronen Shaltiel, and Avi Wigderson. Near-optimal conversion of hardness into pseudo-randomness. In *Proceedings of the Thirty-Ninth Annual IEEE Symposium on Foundations of Computer Science*, New York City, New York, 17-19 October 1999. (To appear.)
22. Russell Impagliazzo and Avi Wigderson. $P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 220–229, El Paso, Texas, 4–6 May 1997.
23. Erich Kaltofen. Polynomial factorization 1987-1991. In I. Simon, editor, *Proc. LATIN '92*, volume 583 of Lect. Notes Comput. Sci., pages 294-313, Heidelberg, Germany, 1992. Springer Verlag.
24. S. Ravi Kumar and D. Sivakumar. Proofs, codes, and polynomial-time reducibilities. In *Proceedings of the Fourteenth Annual IEEE Conference on Computational Complexity*, Atlanta, Georgia, 4-6 May 1999.
25. Richard Lipton. New directions in testing. In *Proceedings of DIMACS Workshop on Distributed Computing and Cryptography*, 1989.
26. Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.
27. Vera Pless, W. Cary Huffman, and Richard A. Brualdi (Eds.). *Handbook of Coding Theory*, North-Holland, 1998.
28. Ran Raz, Omer Reingold, and Salil Vadhan. Extracting all the randomness and reducing the error in Trevisan's extractors. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, Atlanta, GA, May 1999.
29. M. Amin Shokrollahi and Hal Wasserman. List decoding of algebraic-geometric codes *IEEE Transactions on Information Theory*, 45:432–437, March 1999.
30. D. Sivakumar. On membership comparable sets. *Journal of Computer and System Sciences*, 59(2): 270–280, October 1999.
31. Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, March 1997.
32. Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma [extended abstract]. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 537–546, Atlanta, Georgia, 1-4 May 1999.
33. Amnon Ta-Shma and David Zuckerman. Personal communication, November 1999.
34. Luca Trevisan. Construction of extractors using pseudorandom generators. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, Atlanta, Georgia, 1-4 May 1999.

Approximation Algorithms for String Folding Problems

Giancarlo Mauri and Giulio Pavesi

Dept. of Computer Science, Systems and Communication
University of Milan–Bicocca
Milan, Italy
{mauri,pavesi}@disco.unimib.it

Abstract. We present polynomial–time approximation algorithms for string folding problems over any finite alphabet. Our idea is the following: describe a class of feasible solutions by means of an ambiguous context-free grammar (i.e. there is a bijection between the set of parse trees and a subset of possible embeddings of the string); give a score to every production of the grammar, so that the total score of every parse tree (the sum of the scores of the productions of the tree) equals the score of the corresponding structure; apply a parsing algorithm to find the parse tree with the highest score, corresponding to the configuration with highest score among those generated by the grammar. Furthermore, we show how the same approach can be extended in order to deal with an infinite alphabet or different goal functions. In each case, we prove that our algorithm guarantees a performance ratio that depends on the size of the alphabet or, in case of an infinite alphabet, on the length of the input string, both for the two and three–dimensional problem. Finally, we show some experimental results for the algorithm, comparing it to other performance–guaranteed approximation algorithms.

1 Introduction

We present performance–guaranteed approximation algorithms for different versions of the string folding problem. The motivation of string folding problems comes mainly from computational biology. One of the greatest challenges for computational biologists nowadays is to determine the three–dimensional native structure of a protein starting from the amino acid sequence that composes it. The problem has been studied from many different viewpoints, and many models have been proposed. Theoretical models are abstractions of the folding process that emphasize the effect of some factors while hiding other aspects. Perhaps the simplest and most studied model is the *two–dimensional hydrophobic–hydrophilic (HP) model* introduced by Dill [1]. In this model, the amino acid residues are grouped in two classes, according to their chemical properties: the hydrophobic, i.e. non–polar, and hydrophilic, i.e. polar. The protein instance can be thus reduced to a binary sequence of H's (meaning hydrophobic) and P's (meaning hydrophilic). Furthermore, to reduce the number of possible configurations, the

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 45–58, 2000.
© Springer-Verlag Berlin Heidelberg 2000

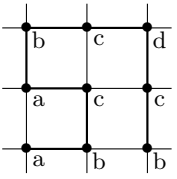


Fig. 1. Two–dimensional embedding of the string abcabcdcb. The score of the embedding is 4.

conformational space is discretized into a square lattice. Feasible structures are therefore mappings (embeddings) of the string into the grid, where adjacent symbols of the string lie on adjacent nodes, and no node is occupied by more than one symbol. It has been observed experimentally that hydrophobic amino acids tend to group inside the native structure, shielded from the environment by the hydrophilic ones. Thus, an optimal configuration for the protein is one that maximizes the number of H's that are *in contact* on the lattice, that is, lie on adjacent nodes of the lattice but are not adjacent in the input string. The study of structures generated by this and other theoretical models can provide useful insights into the dynamics of the folding process [2].

In this paper, we also deal with string folding problems of a more general type. We are given as input a string over some alphabet. The *score* of an embedding of the string is the number of equal symbols of the string that are in contact on the grid. Figure 1 shows an example. Usually, a *neutral symbol* is included in the alphabet. Contacts between neutral symbols do not contribute to the score of an embedding. For example, in the HP model P is the neutral symbol, and the score of the embeddings is determined only by the contacts between H's. The problem is to find the embedding of the string with the maximum score. The three–dimensional version of the problem is defined in the same way; in this case, strings are mapped into the three–dimensional rectangular grid.

The string folding problem over any alphabet (finite or infinite) is NP–hard both in the two and three–dimensional case [3,4,5]. Moreover, the three–dimensional version has been proved to be MAX–SNP hard [6].

The algorithms we present are suitable for more specialized discrete models of the folding of biological sequences, where the goal function does not depend only on contacts between equal symbols, or where contacts between equal symbols have different weights. For example, they could be applied to the string folding problem over an alphabet of twenty symbols, representing the twenty amino acids that build proteins, or to the RNA folding problem over an alphabet of four symbols. Although approximation algorithms for the HP model have already been proposed [7,8], to our knowledge these are the first performance–guaranteed algorithms for the generalized problems. Moreover, the same approach could be easily extended to other discrete or non–discrete models, where the goal function does not necessarily depend on the contacts between equal symbols, as long as the correspondence between parse trees and feasible structures is preserved.

2 The Algorithm

We will now show the basic algorithm, that works on any finite alphabet. Let $\mathbf{\Sigma}$ be the alphabet, with $|\mathbf{\Sigma}| = k$, let a_i, $0 \leq i \leq k-1$ be the symbols of the alphabet. Let a_0 be the neutral symbol, included in the alphabet. Our algorithm is based on the following steps:

1. Define an ambiguous context–free grammar, that generates all the possible instances of the problem (i.e., every string belonging to $\mathbf{\Sigma}^*$).
2. Define a relation between the derivations of the grammar and a subset of all the possible embeddings, where every production of a derivation recursively corresponds to a layout on the lattice of the terminal symbols generated by the production itself.
3. Assign to every production of the grammar an appropriate *score*, representing (a lower bound to) the number of contacts between equal (but not neutral) symbols generated by the spatial position of the symbols associated with the production.
4. Given an instance of the problem, apply a parsing algorithm in order to find the parse tree with the highest score (computed as the sum of the scores of the productions of the tree), that is, the tree corresponding to the embedding of maximum score among those that can be generated by the grammar.

Let us now introduce the grammar we employed in our algorithm. We defined a context–free grammar $G = \{T, N, S, P\}$, where:

1. $T = \{\mathbf{\Sigma} \cup u\}$ is the set of *terminal symbols*, where u is a dummy terminal symbol whose function will be explained later.
2. $N = \{S, L, R\}$ is the set of *nonterminal symbols*.
3. R is the *start symbol*, i.e. the root of every parse tree.
4. P is the set of the productions, composed by the following production schemes:
   (1) $S \rightarrow t_1\ S\ t_2$
   (2) $S \rightarrow t_1\ L\ t_2\ S\ t_3\ L\ t_4$
   (3) $S \rightarrow t_1\ L\ t_2\ S\ t_3 t_4$
   (4) $S \rightarrow t_1 t_2\ S\ t_3\ L\ t_4$
   (5) $S \rightarrow t_1 t_2$
   (6) $S \rightarrow t_1\ L\ t_2 t_3\ L\ t_4$
   (7) $S \rightarrow t_1 t_2 t_3\ L\ t_4$
   (8) $S \rightarrow t_1\ L\ t_2 t_3 t_4$
   (9) $L \rightarrow t_1\ L\ t_2$
   (10) $L \rightarrow t_1 t_2$
   with $t_i \in \mathbf{\Sigma}$;
   and by the following productions, that do not involve symbols from $\mathbf{\Sigma}$:
   (11) $S \rightarrow Suu$
   (12) $S \rightarrow uu$
   (13) $R \rightarrow SS$

The layout of the terminal symbols associated with each production is shown in Fig. 2. The proof that every parse tree corresponds to a feasible structure is straightforward. The score of every production is increased by one every time two equal non–neutral symbols generated by the production are in contact. For example, the production $S \rightarrow a_1 L a_1 S a_1 L a_1$ has score four, $S \rightarrow a_1 L a_1 S a_0 L a_0$ has score one, since neutral symbols do not contribute to the score, and so on. Possible contacts between equal symbols generated by different productions cannot be added to the score of the parse tree.

It could be argued that this grammar generates only sequences of *even* length. To solve this problem, and to avoid adding further productions to the grammar, in case of a sequence s of odd length the string actually parsed is $s^* = sa_0$. In fact, it can be proved that the best embedding among those that can be generated by the algorithm for the original sequence s is the structure found by the algorithm for s^*, with the final neutral symbol removed.

The algorithm builds structures in which the sequence is folded onto itself twice (see Fig. 3). The parse tree is split into two sub–trees, whose roots are the two S symbols generated by the start symbol R. The symbols generated by each sub–tree form a structure shaped like an "U", giving an overall configuration similar to a "C". If the length of the string is even, the first and last symbol are always in contact. Terminals generated by S nonterminals form the "backbone" of the structure, while symbols generated by L nonterminals form lateral branches. The introduction of the dummy terminal symbol u allows the grammar to generate a larger set of structures. The string actually parsed (after the possible addition of a neutral symbol) is $s^u = suu$. If the second sub–tree contains only the production $S \rightarrow uu$, the first sub–tree generates the whole sequence, which is again folded once to form a structure shaped like a "U". Without this extension the algorithm would not be able to generate U–shaped structures, with a significant decrease on its performance ratio (take for instance the string PHPPHP in the HP model, whose optimal structure is U–shaped).

3 The Parsing Algorithm

The parsing algorithm is based on the Earley algorithm for context–free grammars [9], and it is similar to the version that computes the Viterbi parse of a string generated by a stochastic grammar proposed by Stölcke [10]. It preserves the worst case time ($O(n^3)$) and space ($O(n^2)$) complexity of the two algorithms.

The Earley parser keeps a set of *states* for each symbol in the input, describing all pending derivations. A state has the form:

$$i \ : \ {}_kX \rightarrow \ \lambda . \mu$$

where X is a nonterminal symbol of the grammar, λ and μ are strings of terminals or nonterminals, such that $X \rightarrow \lambda\mu$ is a production of the grammar, i and k are indices into the input string. The i indicates that the state belongs to the set associated to the i-th symbol of the input string. The k indicates that the nonterminal X has been expanded starting from the k-th symbol of the input,

(1)	$S \to t_1\ S\ t_2$	$\begin{matrix} \vert & \vert \\ t_1 & t_2 \\ \vert & \vert \end{matrix}$	$+1$ if $t_1 = t_2 \neq a_0$
(2)	$S \to t_1\ L\ t_2\ S\ t_3 t_4$	$\begin{matrix} \vert & \vert \\ -\,t_1 & t_4 \\ (L) & \vert \\ -\,t_2 & t_3 \\ \vert & \vert \end{matrix}$	$+1$ if $t_1 = t_2 \neq a_0$ $+1$ if $t_1 = t_4 \neq a_0$ $+1$ if $t_2 = t_3 \neq a_0$
(3)	$S \to t_1\ L\ t_2\ S\ t_3\ L\ t_4$	$\begin{matrix} \vert & \vert \\ -\,t_1 & t_4\,- \\ (L) & (L) \\ -\,t_2 & t_3\,- \\ \vert & \vert \end{matrix}$	$+1$ if $t_1 = t_4 \neq a_0$ $+1$ if $t_1 = t_2 \neq a_0$ $+1$ if $t_2 = t_3 \neq a_0$ $+1$ if $t_3 = t_4 \neq a_0$
(4)	$S \to t_1 t_2\ S\ t_3\ L\ t_4$	$\begin{matrix} \vert & \vert \\ t_1 & t_4\,- \\ \vert & (L) \\ t_2 & t_3\,- \\ \vert & \vert \end{matrix}$	$+1$ if $t_1 = t_4 \neq a_0$ $+1$ if $t_2 = t_3 \neq a_0$ $+1$ if $t_3 = t_4 \neq a_0$
(5)	$S \to t_1 t_2$	$\begin{matrix} \vert & \vert \\ t_1\!-\!t_2 \end{matrix}$	always zero
(6)	$S \to t_1\ L\ t_2 t_3\ L\ t_4$	$\begin{matrix} \vert & \vert \\ -\,t_1 & t_4\,- \\ (L) & (L) \\ -\,t_2\!-\!t_3 & - \end{matrix}$	$+1$ if $t_1 = t_2 \neq a_0$ $+1$ if $t_1 = t_4 \neq a_0$ $+1$ if $t_3 = t_4 \neq a_0$
(7)	$S \to t_1 t_2 t_3\ L\ t_4$	$\begin{matrix} \vert & \vert \\ t_1 & t_4\,- \\ \vert & (L) \\ t_2\!-\!t_3 & - \end{matrix}$	$+1$ if $t_3 = t_4 \neq a_0$ $+1$ if $t_1 = t_4 \neq a_0$
(8)	$S \to t_1\ L\ t_2 t_3 t_4$	$\begin{matrix} \vert & \vert \\ -\,t_1 & t_4 \\ (L) & \vert \\ -\,t_2\!-\!t_3 \end{matrix}$	$+1$ if $t_1 = t_2 \neq a_0$ $+1$ if $t_1 = t_4 \neq a_0$
(9)	$L \to t_1\ L\ t_2$	$\begin{matrix} -t_1- \\ \\ -t_2- \end{matrix}$	$+1$ if $t_1 = t_2 \neq a_0$
(10)	$L \to t_1 t_2$	$\begin{matrix} t_1- & -t_2 \\ \vert \quad \text{or} \quad \vert \\ t_2- & -t_1 \end{matrix}$	always zero

Fig. 2. Production schemes and corresponding layout of the symbols and scores. a_0 is the neutral symbol.

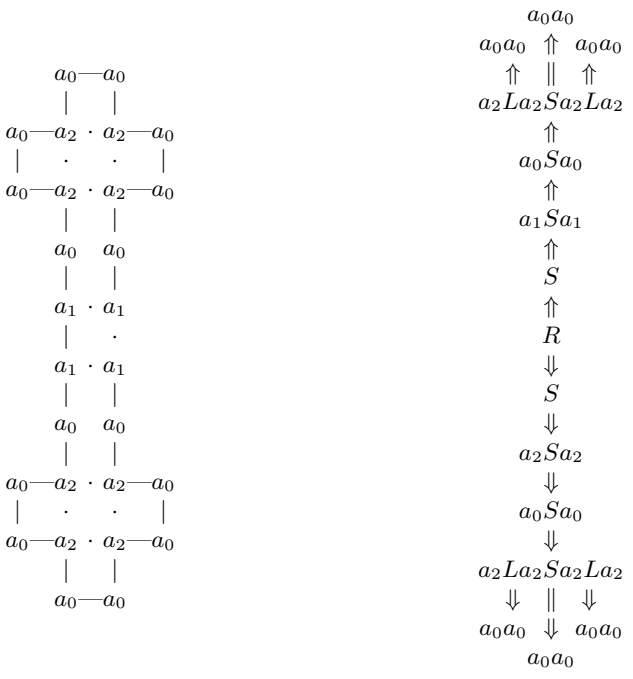


Fig. 3. Structure generated by the algorithm (score 11) for the sequence $a_1a_0a_2a_0a_0a_2a_0a_0a_2a_0a_0a_2a_0a_1a_1a_0a_2a_0a_0a_2a_0a_0a_2a_0a_0a_2a_0a_1$ and corresponding parse tree. Contacts between equal symbols are shown by dots ($\cdot$). $\mathbf{\Sigma} = \{a_0, a_1, a_2\}$, where a_0 is the neutral symbol that does not contribute to the score of the embedding.

and the right–hand side of the production has been expanded up to the position indicated by the dot. A state with the dot at the end of the right–hand side is called a *complete* state, since the dot indicates that the left–hand nonterminal has been completely expanded.

The algorithm is based on three steps that scan the input string from left to right and build new states starting from the current set of states and the current input symbol. The three steps, given in input a string $s = s_0 \dots s_{n-1}$, work as follows.

Prediction For each state

$$i \ : \ {}_kX \rightarrow \ \lambda . Y \mu$$

where Y is a nonterminal, and for all the productions $Y \rightarrow \nu$ of the grammar, add the state:

$$i \ : \ {}_iY \rightarrow . \nu$$

It can be seen that every prediction corresponds to a potential expansion of a nonterminal in a left–most derivation. A state generated by this step is called a *predicted* state.

Scanning For each state

$$i \ : \ {}_kX \rightarrow \ \lambda.a\mu$$

where a is a terminal symbol that matches the current input symbol s_i, add the state:

$$i+1 \ : \ {}_kX \rightarrow \ \lambda a.\mu$$

that is, move the dot one position to the right in the right–hand side. This ensures that terminals generated by the productions match the input string.

Completion For each *complete* state:

$$i \ : \ {}_jY \rightarrow \ \nu.$$

and for each state in the set $j \leq i$

$$j \ : \ {}_kX \rightarrow \ \lambda.Y\mu$$

with the nonterminal Y after the dot, add the state:

$$i \ : \ {}_kX \rightarrow \ \lambda Y.\mu$$

A state generated by the completion step is called a *completed* state. A completed state corresponds to the fact that one of the nonterminal symbols in the right–hand side has been completely expanded (starting from a prediction step) and has generated a sub–string of the input string. The algorithm performs the three operations described above exhaustively, that is, until no new states can be generated.

The algorithm starts from an initial *dummy* state, whose left–hand side is empty:

$$0 \ : \ {}_0 \ \rightarrow .R$$

(R is the start symbol of the grammar). Then, the states corresponding to R (and the possible states deriving from the productions with R on the left–hand side, and so on) are predicted, and the first scanning step examines the first symbol of the input string.

After scanning the last symbol of the string, and performing the corresponding completion step, the algorithm checks whether the state

$$n \ : \ {}_0 \ \rightarrow R.$$

is contained in the last set of states that has been produced, where n is the length of the input string. This means that the start symbol R has been completely expanded in order to build the input string, that is, the string belongs to the language generated by the grammar. If during the computation any set of states remains empty, the algorithm aborts, indicating that a prefix of the input string that cannot be generated by the grammar has been detected.

The only difference between our grammar and a context–free grammar is the introduction of a *score* associated with each production. We modified Earley's algorithm in order to compute the derivation that generates the input string with the highest score. Basically, we added to each state a score, as follows:

$$\begin{gathered} i \ : \ {}_kX \rightarrow \ \lambda.\mu \\ \text{Score} = p \end{gathered}$$

Intuitively, the idea is to have at the end of the parsing a state of the form:

$$\begin{gathered} n \ : \ {}_0 \ \rightarrow \ R. \\ \text{Score} = h \end{gathered}$$

where h is the score corresponding to the highest–score parse tree among those that can be generated for the input string. In order to obtain this result, we modified the three steps of the algorithm as follows.

Prediction For each state

$$\begin{gathered} i \ : \ {}_kX \rightarrow \ \lambda.Y\mu \\ \text{Score} = p \end{gathered}$$

where Y is a nonterminal, and for all the productions $Y \rightarrow \nu$ of the grammar, add the state:

$$\begin{gathered} i \ : \ {}_iY \rightarrow .\nu \\ \text{Score} = 0 \end{gathered}$$

that is, all predicted states have their score set to zero.

Scanning For each state

$$\begin{gathered} i \ : \ {}_kX \rightarrow \ \lambda.a\mu \\ \text{Score} = p \end{gathered}$$

where a is a terminal symbol that matches the current input symbol s_i, add the state:

$$\begin{gathered} i+1 \ : \ {}_kX \rightarrow \ \lambda a.\mu \\ \text{Score} = p \end{gathered}$$

that is, scores are left unchanged by the scanning step.

Completion The actual update of the scores takes place during the completion step. The score of the *complete* states changes as follows. For each *complete* state:

$$\begin{array}{c} i \ : \ {}_jY \rightarrow \ \nu. \\ \text{Score} = p \end{array}$$

the score becomes:

$$\text{Score} = p + q$$

where q is the score associated to the production $Y \rightarrow \nu$. That is, we add to the score of the state (corresponding, as we will see, to the score of the structures generated by the nonterminals contained in the right–hand side of the production) the score corresponding to the layout of the terminal symbols generated by the production itself. Then, for each subset of the current set containing the states $S_1 \ldots S_m$ of the form:

$$\begin{array}{c} i \ : \ {}_jY \rightarrow \ \nu. \\ \text{Score} = q_l \end{array}$$

that is, a subset containing complete states with the same nonterminal on the left–hand side that were predicted at the same j, and for each state:

$$\begin{array}{c} j \ : \ {}_kX \rightarrow \ \lambda.Y\mu \\ \text{Score} = p \end{array}$$

add the state:

$$\begin{array}{c} i \ : \ {}_kX \rightarrow \ \lambda Y.\mu \\ \text{Score} = p + q^* \end{array}$$

where $q^* = \max_{1 \leq l \leq m}\{q_l\}$. That is, we add to the score of the state the score corresponding to the expansion of the nonterminal symbol Y *with the highest score*, i.e. the parse sub–tree with root Y with the highest score. It can be proved recursively that, at the end, the algorithm will associate the start symbol R with the score of the parse tree with the highest score. Moreover, the best parse tree can be reconstructed by assigning to each expanded nonterminal symbol of a completed state the corresponding complete state with the highest score.

4 Dealing with an Infinite Alphabet

In case of an infinite alphabet, the basic algorithm cannot be applied, since it would be impossible to generate a priori all the productions of the grammar and their corresponding scores. The solution we adopted is the following. We let the parser work only on the *production schemes*, and we build the productions on the fly while scanning the input string. The productions contained in the states of the parser have the terminal symbols specified only on the *left* side of the dot. That is, states generated during the prediction step of the parser contain,

instead of terminal symbols, a special wildcard symbol ($*$). Wildcard symbols are replaced by terminals during the scanning step. For example, suppose we are scanning a given symbol s_i of the input string. Each state with a wildcard symbol after the dot:

$$i \ : \ {}_kX \rightarrow \lambda . * \mu$$

generates a new state:

$$i+1 \ : \ {}_kX \rightarrow \lambda s_i . \mu$$

where λ is composed by terminal or nonterminal symbols of the grammar, and μ is composed *only* by wildcard or nonterminal symbols (it does not contain symbols belonging to $\mathbf{\Sigma}$, since it still has to be expanded).

Also, the score of the productions cannot be computed in advance. As we have seen, the score of a state is set to zero, until the state is *completed* or *complete*. For complete states, the score of the corresponding production is computed according to the rules shown in Fig. 2, and then added to the score of the state itself, as in the finite case. When a state is completed, i.e. a nonterminal in its right–hand side has been completely expanded, the score is updated by adding the score of the maximum parse sub–tree generated, once again as in the finite case. Moreover, the scoring scheme for the productions can be easily changed in order to deal with different goal functions.

5 Performance Results

In this section, we prove some results concerning the performance of the algorithm when applied to the different versions of the problem.

In order to have a *performance–guaranteed approximation algorithm*, for every possible instance of the problem the ratio between the score of the structure generated by the algorithm and the score of the optimal structure must be bounded by a constant. That is, for every possible sequence s of arbitrary length we must have:

$$\mathcal{R}(s) = \frac{A(s)}{OPT(s)} \ \geq \mathcal{R} \tag{1}$$

where $A(s)$ is the score of the structure generated by the algorithm when given as input the sequence s, and $OPT(s)$ is the score of the optimal embedding. We will call $\mathcal{R}$ the *absolute performance ratio* of the algorithm, and denote with $\mathcal{R}_k$ the absolute performance ratio when dealing with an alphabet of size k.

5.1 Binary Alphabet

We will start from the performance ratio of the algorithm over a binary alphabet, i.e., in the HP model. Given a string $s = s_0 \ldots s_n$, where $s_i \in \{\mathrm{H}, \mathrm{P}\}$, two symbols s_i and s_j can be in contact on the grid only if $|j - i|$ is odd. Furthermore, every symbol can be in contact with at most two other symbols, except when it is

located at one of the endpoints of the sequence. In this case, it can be in contact with three other symbols.

Now, let h_e be the number of H's in even positions in a given sequence s; h_o the number of H's in odd positions; $h^* = \min\{h_e, h_o\}$. We also define $OPT(s)$ as the score (the number of contacts between H's) of the optimal embedding for a given sequence s. The above considerations yield the following:

Theorem 1.

$$OPT(s) \leq 2h^* + 2 \tag{2}$$

It can be observed that the upper bound $2h^* + 2$ can be reached only by sequences of odd length with two H's at the endpoints. We also can give a lower bound on the number of contacts that are generated by the algorithm.

Lemma 1. *Given a sequence s, there always exists an embedding for s, corresponding to a parse tree generated by the algorithm, that contains $\lceil \frac{h^*+1}{2} \rceil$ contacts between H's.*

The proof of this lemma is quite cumbersome. Actually, we have been able to prove that, in the set of the structures that can be generated by the algorithm, there always exists a structure with $\lceil \frac{h^*+1}{2} \rceil$ contacts, but not, for example, that this is the best structure of the set. Thus, we could give only a lower bound on the actual performance ratio of the algorithm. In fact, as shown in Section 6, the worst case that we found experimentally gave a performance ratio of 3/8. This will also affect, as we will see, the performance ratio for the more general versions. From the result of Lemma 1, however, it is straightforward to obtain the performance ratio of the algorithm.

Theorem 2.

$$\mathcal{R}_2 \geq \frac{\lceil \frac{h^*+1}{2} \rceil}{2h^* + 2} \geq \frac{1}{4} \tag{3}$$

5.2 Finite Alphabet

We will now show the results concerning the performance of the algorithm applied to the problem over any finite alphabet. Let s be a string of symbols taken from an alphabet $\mathbf{\Sigma}$, with $|\mathbf{\Sigma}| = k$. Also, let σ_i^o and σ_i^e, mbox1 $\leq i \leq k-1$ be the number of the occurrences of the symbol $a_i \in \mathbf{\Sigma}$ in the sequence s respectively in an odd and in an even position (we do not consider the number of occurrences of the neutral symbol a_0). Finally, let $\sigma_i^* = \min\{\sigma_i^o, \sigma_i^e\}$, $1 \leq i \leq k-1$, and $\sigma^* = \max_{1 \leq i \leq k}\{\sigma_i^*\}$.

Lemma 2. *For any sequence s over an alphabet $\mathbf{\Sigma}$ with $|\mathbf{\Sigma}| = k$,*

$$OPT(s) \leq 2 \sum_{i=1}^{k-1} \sigma_i^* + 2 \tag{4}$$

By considering the symbol that corresponds to σ^* the only non–neutral symbol of the alphabet, we can easily prove the following lemma.

Lemma 3. *For any sequence s over a finite alphabet, the set of structures generated by the algorithm contains a structure that generates at least $\lceil \frac{\sigma^*+1}{2} \rceil$ contacts.*

We want to point out that once again the structure of Lemma 3 does not correspond to the best structure that can be generated by the algorithm. In fact, the algorithm tries to generate contacts not only with the symbol corresponding to σ^*, but with all the non–neutral symbols, as shown in Fig. 3. Lemma 3 guarantees only three contacts between a_2 symbols, while the solution found by the algorithm contains eleven contacts, and corresponds to the optimal embedding. However, starting from the previous two lemmas, the worst case is a sequence where the number of occurrences of every non–neutral symbol is equal. Therefore, we have $OPT(s) \leq 2\sigma^*(k-1)+2$. This fact yields the following theorem:

Theorem 3.

$$\mathcal{R}_k \geq \frac{\lceil \frac{\sigma^*+1}{2} \rceil}{2\sum_{i=1}^{k-1} \sigma_i^* + 2} \geq \frac{\sigma^*+1}{4[\sigma^*(k-1)+1]} \geq \frac{1}{4(k-1)} \tag{5}$$

5.3 Infinite Alphabet

For the proof of the performance ratio of the algorithm applied to the problem over an infinite alphabet, we start from the fact that, although the size of the alphabet is not bounded, the input sequence is finite. Therefore, the number of different symbols occurring in the sequence that can generate contacts is also finite. Let d be this number. The terms σ_i^* and σ^* are defined as in the previous section, but in this case we have $1 \leq i \leq d$, with $d \leq n/2$. Thus, the performance ratio of the algorithm on a given input s of length n can be defined as follows.

Theorem 4.

$$\mathcal{R}_\infty \geq \frac{\lceil \frac{\sigma^*+1}{2} \rceil}{2\sum_{i=1}^{d} \sigma_i^* + 2} \geq \frac{\sigma^*+1}{4[\sigma^*(\frac{n}{2}-1)+1]} \geq \frac{1}{4(\frac{n}{2}-1)} \tag{6}$$

5.4 The 3D Case

Although structures generated by our algorithm are two–dimensional, they anyway guarantee a performance ratio also for the three-dimensional problem. In the three-dimensional lattice, each non–neutral symbol can be in contact with at most four equal symbols (five, if it is located at the endpoints of the sequence). This fact yields the following lemma.

Lemma 4. *For any sequence s over an alphabet $\mathbf{\Sigma}$ with $|\mathbf{\Sigma}| = k$,*

$$OPT(s) \leq 4\sum_{i=1}^{k-1} \sigma_i^* + 2 \tag{7}$$

This, together with Lemma 3, gives the absolute performance ratio $\mathcal{R}_k^{3d}$ for the three-dimensional problem over an alphabet of size k.

Theorem 5.

$$\mathcal{R}_k^{3d} \geq \frac{\lceil \frac{\sigma^*+1}{2} \rceil}{4\sum_{i=1}^{k-1} \sigma_i^* + 2} \geq \frac{\sigma^*+1}{8[\sigma^*(k-1)+1]} \geq \frac{1}{8(k-1)} \tag{8}$$

It is worth mentioning that in the case of a binary alphabet the absolute performance ratio of our algorithm (1/8) equals the best approximation algorithm known for the three–dimensional problem [7].

6 Experimental Evaluation

In the two–dimensional binary case, our algorithm equals the performance ratio of the best algorithms so far proposed [7]. Therefore, we have tested it on random instances of the two–dimensional problem over a binary alphabet, and compared the results to the other algorithms (see Table 1). Given $P_H = \Pr[s_i = H]$, $\forall i \in [0, n]$, for different values of P_H we have completed 1000 runs of our algorithm and of the other two with performance–guaranteed ratios of 1/4 (called $\mathcal{B}$ and $\mathcal{C}$ as in the original paper), on instances of length 63 with two H's at the endpoints (in order to reach the higher bound for the goal function).

The performance ratio of our algorithm seems to decrease as the average number of H's in the sequence is increased. The same trend, even if with lower ratios, is shown by algorithm $\mathcal{C}$, while algorithm $\mathcal{B}$ has a constant ratio. In the tests, the worst case ratio of 1/4 has been reached only by algorithm $\mathcal{B}$, while on sequences like $\mathrm{PP(HPP)}^{4k+1}$, $k \geq 3$ (whose optimal score is $4k$), algorithm $\mathcal{C}$ produces structures with score $k+3$. Thus, its performance ratio approaches 1/4 as k is increased [7]. It should be noted that our algorithm found the optimal

Table 1. Average performance ratios of algorithms $\mathcal{B}$ and $\mathcal{C}$ [7], and our algorithm (CFG) in the two–dimensional case of the problem over a binary alphabet, for different values of $P_H = \Pr[s_i = H], \forall i \in [0, n]$.

Algorithm	$\mathcal{B}$	$\mathcal{C}$	CFG
$P_H = .15$	0.52	0.60	0.79
$P_H = .33$	0.48	0.57	0.72
$P_H = .5$	0.48	0.55	0.68
$P_H = .66$	0.48	0.53	0.63
$P_H = .85$	0.48	0.50	0.55
Average	0.48	0.55	0.67
Worst case	0.25	0.33	0.375

solution on this set of instances. The worst case found for our algorithm is 3/8: this, as discussed in the previous sections, leaves open the issue of its performance ratio.

7 Conclusions

We presented polynomial–time approximation algorithms for the string folding problem that guarantee performance ratios both for the two– and three–dimensional case, and work over any alphabet, finite and infinite. To our knowledge, these are the first performance–guaranteed approximation algorithms that can be applied to the problem over alphabets larger than the binary one, while for the latter we equaled the performances of the best algorithms so far proposed, with better experimental results. In each case, the performance ratios that have been proved serve only as a lower bound for the actual ones. Our approach can also be easily extended to different goal functions, and also to more powerful grammars and non–discrete versions of string folding problems, as long as the correspondence between parse trees and feasible structures is preserved.

Acknowledgements

This work has been supported by the Italian Ministry of University, under the project "Bioinformatics and Genomic Research".

References

1. K.A. Dill, Dominant forces in protein folding. Biochemistry, **24**(1985), 1501.
2. B. Hayes, Prototeins. American Scientist, **3**(1998), 86.
3. M. Paterson, T. Przytycka, On the Complexity of String Folding. Discrete and Applied Maths, **71**(1996), 217–230.
4. P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, M. Yannakakis, On the Complexity of Protein Folding. Proc. of the Second Annual International Conference on Computational Biology (RECOMB '98), 61–62, New York, 1998.
5. B. Berger, T. Leighton, Protein Folding in the HP Model is NP Complete. Proc. of the Second Annual International Conference on Computational Biology (RECOMB '98), 30–39, New York, 1998.
6. A. Nayak, A. Sinclair, U. Zwick, Spatial Codes and the Hardness of String Folding Problems. Proceedings of the 9^{th} Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '98), 639–648, San Francisco, 1998.
7. W.E. Hart, S.C. Istrail, Fast Protein Folding in the Hydrophobic-Hydrophilic Model. Journal of computational biology, **3**(1), 53–96, 1996.
8. G. Mauri, G. Pavesi, A. Piccolboni, Approximation Algorithms for Protein Folding Prediction. Proceedings of the 10^{th} Annual ACM–SIAM Symposium on Discrete Algorithms (SODA '99), S945–S946, Baltimore, 1999.
9. J. Earley, An Efficient Context-Free Parsing Algorithm. Communications of the ACM, **6**(1970), 451–455.
10. A. Stölcke, An Efficient Probabilistic Context–Free Parsing Algorithm That Computes Prefix Probabilities. Computational Linguistics, **21**(2), 165–201, 1995.

An Index for Two Dimensional String Matching Allowing Rotations

Kimmo Fredriksson[1]*, Gonzalo Navarro[2]**, and Esko Ukkonen[1]***

[1] University of Helsinki, Department of Computer Science,
P.O.Box 26, FIN-00014 Helsinki, Finland,
Fax: +358 9 1914 4441, Kimmo.Fredriksson@cs.Helsinki.FI.
[2] Department of Computer Science, University of Chile.

Abstract. We present an index to search a two-dimensional pattern of size $m \times m$ in a two-dimensional text of size $n \times n$, even when the pattern appears rotated in the text. The index is based on (path compressed) tries. By using $O(n^2)$ (i.e. linear) space the index can search the pattern in $O((\log_\sigma n)^{5/2})$ time on average, where σ is the alphabet size. We also consider various schemes for approximate matching, for which we obtain either $O(\text{polylog}_\sigma n)$ or $O(n^{2\lambda})$ search time, where $\lambda < 1$ in most useful cases. A larger index of size $O(n^2(\log_\sigma n)^{3/2})$ yields an average time of $O(\log_\sigma n)$ for the simplest matching model. The algorithms have applications e.g. in content based information retrieval from image databases.

1 Introduction

Two dimensional pattern (image) matching has important applications in many areas, ranging from science to multimedia. String matching is one of the most successful special areas of algorithmics. Its theory and potential applications in the case of one–dimensional data, that is, linear strings and sequences, is well understood. However, the string matching approach still has considerable unexplored potential when the data is more complicated than just a linear string. Two dimensional digital images are an example of such a data.

Examples of combinatorial pattern matching algorithms that work in two dimensions, but do not allow rotations are e.g. [1,7,11,12,13,14,15]. On the other hand, there are many non–combinatorial approaches to rotation invariant pattern matching, for a review, see e.g. [16]. The only combinatorial methods, that come close to us in some respects, are [12,14]. However, these do not address the pattern rotations. As stated in [2], a major open problem in two–dimensional (combinatorial) pattern matching is to find the occurrences of a two-dimensional

* Work supported by ComBi.
** Work developed while the author was in a postdoctoral stay at the Dept. of Computer Science, Univ. of Helsinki. Partially supported by the Academy of Finland and Fundación Andes.
*** Work supported by the Academy of Finland.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 59–75, 2000.
© Springer-Verlag Berlin Heidelberg 2000

pattern of size $m \times m$ in a two-dimensional text of size $n \times n$ when the pattern can appear in the text in rotated form. This was addressed from a combinatorial point of view first in [3], in which an online algorithm allowing pattern rotations was presented.

In this work we give the first algorithms for offline searching, that is, for building an index over the text that allows fast querying. The data structure we use is based on tries. Suffix trees for two-dimensional texts have been considered, e.g. in [8,9,10]. The idea of searching a rotated pattern using a "suffix" array of spiral like strings is mentioned in [10], but only for rotations of multiples of 90 degrees. The problem is much more complex if we want to allow any rotation.

In [3] the consideration was restricted to the matches of a pattern inside the text such that the geometric center of the pattern has been put exactly on top of the exact center point of some text cell. This is called the "center–to–center assumption". Under this assumption, there are $O(m^3)$ different relevant rotation angles to be examined for each text cell. In this paper we make this assumption, too, and consider the following four matching models:

Exact: the value of each text cell whose center is covered by some pattern cell must match the value of the covering pattern cell.

Hamming: an extension of the Exact model in which an error threshold $0 \leq k < m^2$ is given and one is required to report all text positions and rotation angles such that at most k text cells do not match the covering pattern cell.

Grays: an extension of the Exact model more suitable for gray level images: the value of each text cell involved in a match must be between the minimum and maximum value of the 9 neighboring cells surrounding the corresponding pattern cell [5].

Accumulated: an extension of the Hamming model, more suitable for gray levels. The sum of the absolute differences between the value of the text cells involved in a match and the values of the corresponding patterns cells must not exceed a given threshold k.

Our results are summarized in Table 1. For some algorithms we have two versions, one with pattern partitioning technique, and one without it. We denote by σ the alphabet size and assume in our average case results that the cell values are uniformly and independently distributed over those σ values. The times reported are average–case bounds for the search. In the Hamming and Accumulated models $\alpha = k/m^2$ (note that $\alpha < 1$ for Hamming and $\alpha < \sigma$ for Accumulated) and k^*_H and k^*_A denote the maximum k values up to where some techniques work: $k^*_H = k/(1 - e/\sigma)$ and $k^*_A = k/(\sigma/(2e) - 1)$. Moreover, $H^H_\sigma(\alpha) = -\alpha \log_\sigma(\alpha) - (1-\alpha)\log_\sigma(1-\alpha)$ and $H^A_\sigma(\alpha) = -\alpha \log_\sigma(\alpha) + (1+\alpha)\log_\sigma(1+\alpha)$. According to Table 1, the search times are sublinear on average when the conditions are met, which implies in particular that $\alpha < 1 - e/\sigma$ for Hamming and $\alpha < \sigma/(2e) - 1$ for Accumulated. In all the cases the index needs $O(n^2)$ space and it can be constructed in average time $O(n^2 \log_\sigma n)$.

We have also considered the alternative model in which the pattern centers are used instead of the text centers. For this case we obtain an index that for the Exact model needs $O(n^2(\log_\sigma n)^{3/2})$ space and gives $O(\log_\sigma n)$ time.

Model	Search time	Condition
Exact	$(\log_\sigma n)^{5/2}$	$\pi m^2/4 \geq \log_\sigma n^2$
Hamming	$(2\log_\sigma n)^{k+3/2}(\sigma/k)^k$	$\pi m^2/4 \geq \log_\sigma n^2 > k^*_H$
Hamming (pattern partitioning)	$n^{2(\alpha+H^H_\sigma(\alpha))}m^5/\log_\sigma n$	$\pi m^2/4 \geq \log_\sigma n^2 > k^*_H$
Grays	$n^{2(1-\log_\sigma(5/4))}(\log_\sigma n)^{3/2}$	$\pi m^2/4 \geq \log_\sigma n^2$
Accumulated	$n^{\log_\sigma 4}(1+2(\log_\sigma n)/k)^k(\log_\sigma n)^{3/2}$	$\pi m^2/4 \geq \log_\sigma n^2 > k^*_A$
Accumulated (pattern partitioning)	$n^{2(\log_\sigma 2+H^A_\sigma(\alpha))}m^5/\log_\sigma n$	$\pi m^2/4 \geq \log_\sigma n^2 > k^*_A$

Table 1. Time complexities achieved under different models.

The algorithms are easily generalized for handling large databases of images. That is, we may store any number of images in the index, and search the query pattern simultaneously from all the images. The time complexities remain the same, if we now consider that n^2 denotes the size of the whole image library.

2 The Data Structures

Let $T = T[1..n, 1..n]$ and $P = P[1..m, 1..m]$ be two dimensional arrays of *point samples*, such that $m < n$. Each sample has a *color* in a finite ordered *alphabet* Σ. The size $|\Sigma|$ of Σ is denoted by σ. The arrays P and T are point samples of colors of some "natural" image. There are several possibilities to define a mapping between T and P, that is, how to compare the colors of P to colors of T. Our approach to the problem is combinatorial. Assume that P has been put on top of T, in some arbitrary position. Then we will compare each color sample of T against the color of the closest sample of P. The distance between the samples is simply the Euclidean distance. This is also technically convenient. The Voronoi diagram for the samples is a regular array of unit squares.

Hence we may define that the array T consists of n^2 unit squares called *cells*, in the real plane $\mathbf{R}^2$ (the (x, y)–plane). The corners of the cell for $T[i, j]$ are $(i-1, j-1), (i, j-1), (i-1, j)$ and (i, j). Each cell has a *center* which is the geometric center point of the cell, i.e., the center of the cell for $T[i, j]$ is $(i-\frac{1}{2}, j-\frac{1}{2})$. The array of cells for pattern P is defined similarly. The *center* of the whole pattern P is the center of the cell in the middle of P. Precisely, assuming for simplicity that m is odd, the center of P is the center of cell $P[\frac{m+1}{2}, \frac{m+1}{2}]$. For images, the cells are usually called *pixels*.

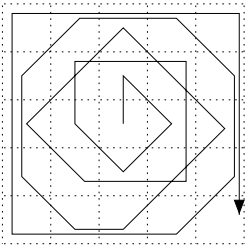
Assume now that P has been moved on top of T using a rigid motion (translation and rotation), such that the center of P coincides exactly with the center of some cell of T. The location of P with respect to T can be uniquely given as $((i-\frac{1}{2}, j-\frac{1}{2}), \theta)$, where $(i-\frac{1}{2}, j-\frac{1}{2})$ is the location of the center of P in T, and θ is the angle between the x–axis of T and the x–axis of P. The occurrence (or more generally, distance) between T and P at some location, is determined by comparing the colors of the cells of T and P that overlap. We will use the centers of the cells of T for selecting the comparison points. That is, for the pattern at location $((i-\frac{1}{2}, j-\frac{1}{2}), \theta)$, we look which cells of the pattern cover the centers of the cells of the text, and compare the corresponding colors of those cells. As

the pattern rotates, the centers of the cells of the text move from one cell of P to another. In [3] it is shown that this happens $O(m^3)$ times, so there are $O(m^3)$ relevant orientations of P to be checked. The actual comparison result of two colors depends on the matching model.

We propose to use a trie based index of the text, defined as follows. Trie is a well–known tree structure for storing strings in which each edge is labeled by a character. Each cell of the text defines a string which is obtained by reading text positions at increasing distances from the center of the cell. The first character is that of the cell, then come the 4 closest centers (from the cells above, below, left and right of the central cell), then the other 4 neighbors, and so on. The cells at the same distance are read in some predefined order, the only important thing is to read the cells in the order of increasing distance from the center cell. This effectively utilizes the $O(m^3)$ result to restrict the number of rotations our algorithms must consider on average, see Sec. 3. If such a string hits the border of the text it is considered finished there. We will call *sistrings* (for "semi–infinite strings") [10] the strings obtained in this way. Figure 1 shows a possible reading order.

22	16	10	17	23
15	5	1	6	18
9	4	0	2	11
14	8	3	7	19
21	13	12	20	24

(a)



(b)

2	56	12	1	3
1	7	2	5	9
3	6	3	19	1
31	28	2	5	23
2	13	12	22	4

(c)

Fig. 1. A possible reading order ("spiral") for the sistring that starts in the middle of a text of size 5×5. Figure *(a)* shows the reading order by enumerating the cells, and figure *(b)* shows the enumeration graphically. Figure *(c)* shows the color values of the cells of the image, and for that image the sistring corresponding to the reading order is $< 3, 2, 19, 2, 6, 7, 5, 5, 28, 3, 12, 1, 12, 13, 31, 1, 56, 1, 9, 23, 22, 2, 2, 3, 4 >$

Therefore each text cell defines a sistring of length $O(n^2)$. A trie on those strings (called the *sistring trie*) can be built, which has average size $O(n^2)$ and average depth $O(\log_\sigma n^2)$. Alternatively, the unary paths of such a trie can be compressed, in similar manner used to compress the suffix trees. In such a tree each new string adds at most one leaf and one internal node, so the worst case size is $O(n^2)$.

Still another possibility is to construct an array of n^2 pointers to T, sorted in the lexicographic order of the n^2 sistrings in T. Such an array, called the *sistring array*, can be formed by reading the leaves of a sistring trie in the lexicographic order, or directly, by sorting the sistrings. The array needs $O(n^2)$ space, but is

much smaller that the sistring trie or tree in practice. Hence the sistring array is the most attractive alternative from the practical point of view and will therefore be used in the experiments.

The sistring trie can be built in $O(n^2 \log_\sigma n)$ average time, by level–wise construction. The sistring array can be built in $O(n^2 \log n)$ string comparisons, which has to be multiplied by $O(\log_\sigma n^2)$ to obtain the average number of character comparisons. The sistring array is very similar to the suffix array, which in turn is a compact representation of a suffix tree.

For simplicity, we describe the algorithms for the sistring trie, although they run with the same complexity over sistring trees. For sistring arrays one needs to multiply the search time results by $O(\log n)$ as well, because searching the array uses binary search.

We consider now a property of the sistring trie that is important for all the results that follow. We show that under a uniform model, the number of sistring trie nodes at depth ℓ is $\Theta(\min(\sigma^\ell, n^2))$. This roughly is to say that in levels $\ell \leq h$, for $h = \log_\sigma(n^2) = 2\log_\sigma n$ all the different strings of length ℓ exist, while from that level on the $\Theta(n^2)$ sistrings are already different. In particular this means that nodes deeper than h have $O(1)$ children because there exists only one sistring in the text with that prefix of length h (note that a sistring prefix is graphically seen as a spiral inside the text, around the corresponding text cell).

To prove this property we consider that there are n^2 sistrings uniformly distributed across σ^ℓ different prefixes, of length ℓ, for any ℓ. The probability of a prefix not being "hit" after n^2 attempts is $(1-1/\sigma^\ell)^{n^2}$, so the average number of different prefixes hit (i.e. existing sistring trie nodes) is

$$\sigma^\ell(1-(1-1/\sigma^\ell)^{n^2}) \;=\; \sigma^\ell(1-e^{-\Theta(n^2/\sigma^\ell)}) \;=\; \sigma^\ell(1-e^{-x})$$

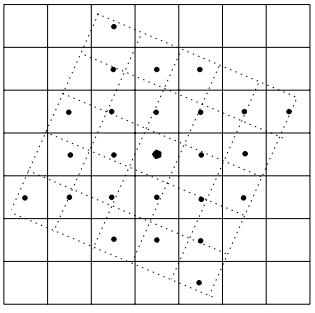
for $x = \Theta(n^2/\sigma^\ell)$. Now, if $n^2 = o(\sigma^\ell)$ then $x = o(1)$ and $1 - e^{-x} = 1 - (1 - x + O(x^2)) = \Theta(x) = \Theta(n^2/\sigma^\ell)$, which gives the result $\Theta(n^2)$. On the other hand, if $n^2 = \Omega(\sigma^\ell)$ then $x = \Omega(1)$ and the result is $\Theta(\sigma^\ell)$. Hence the number of sistring trie nodes at depth ℓ is on average $\Theta(\min(\sigma^\ell, n^2))$, which is the same as the worst case. Indeed, in the worst case the constant is 1, i.e. the number of different strings is at most $\min(\sigma^\ell, n)$, while on average the constant is smaller. We need this result for giving bounds for the maximum number of sistring trie nodes inspected by our algorithms.

3 The Exact Model

We first describe the algorithm for the exact matching model. The other algorithms are relatively straight–forward extensions of it. As shown in [3], there are $O(m^3)$ relevant orientations in which the pattern can occur in a given text position. A brute force approach is to consider the $O(m^3)$ pattern orientations in turn and search each one in the sistring trie. To check the pattern in a given orientation we have to see in which order the pattern cells have to be read so that they match the reading order of the sistring trie construction.

Figure 2 shows the reading order induced in the pattern by a rotated occurrence, using the spiral like reading order given in Figure 1. For each possible rotation we compute the induced reading order, build the string obtained by reading the pattern in that order from its center, and search that string in the sistring trie. Note in particular that some pattern cells may be read twice and others may not be considered at all. Observe that in our example the cells numbered 30, 32, 34, and 36 are outside the maximum circle contained in the pattern, and are therefore ignored in the sistring trie search. This is because those values cannot be used unless some levels are skipped in the search, which would mean entering into all the branches after reading cell number 20. Text cells 21–29, 31, 33, 35, and 37– all fall outside the pattern.

The algorithm first considers the sistring of P for angle $\theta = 0$. The sistring is searched from the trie until some cell of P mismatches, at depth ℓ. Now the pattern must be rotated a little in order to read the next sistring. The next rotation to try is such that any of the first ℓ cells of the previous sistring changes, that is, any of the centers of the cells of T hits some border of the first ℓ sistring cells of P.



(a)

34	10	17	18	*36*
16	5	1	6	11
15	4	0	2	19
9	8	3	7	20
32	14	13	12	*30*

(b)

2	19	7	1	3
1	7	2	5	9
3	6	3	4	1
9	12	2	5	3
2	7	6	5	4

(c)

Fig. 2. Reading order induced in the pattern by a rotated occurrence. Figure *(a)* shows the pattern superimposed on the image, Figure *(b)* shows the enumeration of the induced reading order, and Figure *(c)* shows the color values for the pattern cells. The corresponding sistring is $< 3, 2, 4, 2, 6, 7, 5, 5, 12, 9, 19, 9, 5, 6, 7, 3, 1, 7, 1, 1, 3 >$. Cells numbered 30, 32, 34, and 36 are ignored in the trie search.

The number of rotations to try depends on how far we are from the center. That is, the number of the text centers that any cell of P may cover depends on how far the cell is from the rotation center. If the distance of some cell of P from the rotation center id d, then it may cover $O(d)$ center of T. In general, there are $O(m^3)$ rotations for a pattern of size $m \times m$ cells. The number of rotations grows as we get farther from the center, and they are tried only on the existing branches of the sistring trie. As the pattern is read in a spiral form, when we are at depth ℓ in the sistring trie we are considering a pattern cell which is at distance $O(\sqrt{\ell})$ from the center. This means that we need to consider $O(\ell^{3/2})$

different rotations if the search has reached depth ℓ. For each rotation we assume in the analysis that the sistring is read and compared from the beginning.

The fact that on average every different string up to length h exists in the sistring trie means that we enter always until depth h. The number of sistring trie nodes considered up to depth h is thus

$$\sum_{\ell=1}^{h} \ell^{3/2} \quad = \quad O(h^{5/2})$$

At this point we have $O(h^{3/2})$ candidates that are searched deeper in the sistring trie. Now, each such candidate corresponds to a node of the sistring trie at depth h, which has $O(1)$ children because there exist $O(1)$ text sistrings which share this prefix with the pattern (a "prefix" here means a circle around the pattern center).

Two alternative views for the same process are possible. First, consider all the $O(h^{3/2})$ candidates together as we move to deeper levels $\ell > h$ in the sistring trie. There are on average n^2/σ^ℓ sistrings of length ℓ matching a given string, so the total work done when traversing the deeper levels of the sistring trie until the candidates get eliminated is

$$\sum_{\ell \geq h+1} \frac{n^2}{\sigma^\ell} \ell^{3/2} \quad = \quad \sum_{\ell \geq 1} \frac{(\ell+h)^{3/2}}{\sigma^\ell} \quad = \quad O(h^{3/2})$$

An alternative view is that we directly check in the text each candidate that arrived to depth h, instead of using the sistring trie. There are $O(h^{3/2})$ candidates and each one can be checked in $O(1)$ time: if we perform the comparison in a spiral way, we can add the finer rotations as they become relevant. The ℓ-th pattern cell in spiral order (at distance $\sqrt{\ell}$ from the center) is compared (i.e. the comparison is not abandoned before) with probability $\ell^{3/2}/\sigma^\ell$. Summing up the probabilities of being compared over all the characters yields

$$\sum_{\ell \geq 1} \frac{\ell^{3/2}}{\sigma^\ell} \quad = \quad O(1)$$

where for simplicity we have not considered that we have compared already h characters and have an approximate idea of the orientations to try.

Therefore we have a total average search cost of $O((\log_\sigma n)^{5/2})$. This assumes that the pattern is large enough, i.e. that we can read h characters from the center in spiral form without hitting the border of the pattern. This is equivalent to the condition $m^2 \geq \frac{4}{\pi} \log_\sigma n^2$ which is a precondition for our analysis. Smaller patterns leave us without the help of the sistring trie long before we have eliminated enough candidates to guarantee low enough average time to check their occurrences in the text.

4 The Hamming Model

This model is an extension of the exact matching model. An additional parameter k is provided to the search algorithm, such that a mismatch occurs only when more than k characters have not matched. In this section we use $\alpha = k/m^2$. We require $0 \leq \alpha < 1$, as otherwise the pattern would match everywhere.

The problem is much more complicated now. Even for a fixed rotation, the number of sistring trie nodes to consider grows exponentially with k. To see this, note that at least k characters have to be read in all the sistrings, which gives a minimum of $O(\sigma^k)$ nodes. This means in particular that if $k \geq h$ then we will consider the n^2 sistrings and the index will be of no use, so we assume $k < h$; still stricter conditions will appear later. We first present a standard technique and then a pattern partitioning technique.

4.1 Standard Searching

Imagine that for each possible rotation we backtrack on the sistring trie, entering into all the possible branches and abandoning a path when more than k mismatches have occurred. As explained, up to depth k we enter into all the branches. Since $h > k$, we have to analyze which branches we enter at depths $k < \ell \leq h$. Since all those strings exist in the sistring trie, this is the same as to ask how many different strings of length ℓ match a pattern prefix of length ℓ with at most k mismatches.

A pessimistic model assumes that there are $\binom{\ell}{k}$ ways to choose the cells that will not match, and σ^k selections for them. As we can replace a cells by itself we are already counting the cases with less than k errors. The model is pessimistic because not all these choices lead to different strings. To all this we have to add the fact that we are searching $O(\ell^{3/2})$ different strings at depth ℓ. Hence, the total number of sistring trie nodes touched up to level h is

$$\sum_{\ell=1}^{k} \ell^{3/2}\sigma^\ell \;+\; \sum_{\ell=k+1}^{h} \ell^{3/2}\binom{\ell}{k}\sigma^k \;=\; O\left(h^{3/2}\sigma^k\binom{h}{k}\right)$$

For the second part of the search, we consider that there are on average n^2/σ^ℓ sistrings of length $\ell > h$ equal to a given string. So the total amount of nodes touched after level h is

$$\sum_{\ell \geq h+1} \ell^{3/2}\binom{\ell}{k}\sigma^k\frac{n^2}{\sigma^\ell} \;=\; \sum_{\ell \geq h+1} \ell^{3/2}n^2\binom{\ell}{k}\frac{1}{\sigma^{\ell-k}}$$

In the Appendix A we show that $\binom{\ell}{k}\frac{1}{\sigma^{\ell-k}}$ is exponentially decreasing with ℓ for

$$\ell \;>\; k^*_H \;=\; \frac{k}{1-e/\sigma}$$

while otherwise it is $\Omega(1/\sqrt{\ell})$.

Therefore, the result depends on whether or not $h > k_H^*$. If $h \leq k_H^*$, then the first term of the summation alone is $\Omega(hn^2)$ and there is no sublinearity. If, on the other hand, $h > k_H^*$, we have an exponentially decreasing series where the first term dominates the whole summation. That is, the cost of the search in levels deeper than h is

$$h^{3/2}n^2\binom{h}{k}\frac{1}{\sigma^{h-k}} \quad = \quad h^{3/2}\binom{h}{k}\sigma^k \quad = \quad O\left((\log_\sigma n)^{k+3/2}(\sigma/k)^k\right)$$

which matches the cost of the first part of the search as well. Therefore, the condition for a sublinear search time is $k_H^* < h < m^2$. This in particular implies that $\alpha < 1 - e/\sigma$.

4.2 Pattern Partitioning

The above search time is still polylogarithmic in n, but exponential in k. We present now a *pattern partitioning* technique that obtains a cost of the form $O(n^{2\lambda})$ for $\lambda < 1$. The idea is to split the pattern in j^2 pieces (j divisions across each coordinate). If there are at most k mismatches in a match, then at least one of the pieces must have at most $\lfloor k/j^2 \rfloor$ errors. So the technique is to search for each of the j^2 pieces (of size $(m/j) \times (m/j)$) separately allowing k/j^2 errors, and for each (rotated) match of a piece in the text, go to the text directly and check if the match can be extended to a complete occurrence with k errors. Note that the α for the pieces is the same as for the whole pattern.

The center–to–center assumption does not hold when searching for the pieces. However, for each possible rotation of the whole pattern that matches with the center–to–center assumption, it is possible to fix some position of the center of each piece inside its text cell. (The center of the piece is ambiguous, as there are infinitely many angles for the matching pattern: there are $O(m^3)$ different relevant rotations of the pattern, and between the corresponding angles, there are infinitely many angles where the occurrence status does not change. However, any of the possible positions for the center of the pieces can be chosen). The techniques developed to read the text in rotated form can be easily adapted to introduce a fixed offset at the center of the matching subpattern. Therefore we search each of the j^2 pieces in every of the $O(m^3)$ different rotations.

The search cost for this technique becomes j^2m^3 times the cost to search a piece (with a fixed rotation and center offset) in the sistring trie and the cost to check for a complete occurrence if the piece is found.

If we consider that $(m/j)^2 \leq h$, then all the strings exist when a piece is searched. Therefore the cost to traverse the sistring trie for a piece at a fixed rotation is equivalent to the number of strings that can be obtained with k mismatches from it, i.e.

$$U = \binom{(m/j)^2}{k/j^2}\sigma^{k/j^2}$$

while the cost to check all the U candidates is $Ukn^2/\sigma^{(m/j)^2}$, i.e. k times per generated string times the average number of times such a string appears in the

text. Therefore the overall cost is

$$j^2 m^3 \left(U \ + \ Uk \frac{n^2}{\sigma^{(m/j)^2}} \right)$$

where (after distributing the initial factor) the first term decreases and the second term increases as a function of j. The optimum is found when both terms meet, i.e. $j = m/\sqrt{2 \log_\sigma n}$ which is in the limit of our condition $(m/j)^2 \leq h$. In fact, the second term is decreasing only for $\alpha < 1 - e/\sigma$, otherwise the optimum is $j = 1$, i.e. no pattern partitioning.

For this optimal j, the overall time bound becomes

$$O\left(\frac{m^5}{\log_\sigma n} \ n^{2(\alpha + H_\sigma^H(\alpha))} \right)$$

where we have written $H_\sigma^H(\alpha) = -\alpha \log_\sigma(\alpha) - (1-\alpha) \log_\sigma(1-\alpha)$.

This bound is sublinear as long as $\alpha < 1 - e/\sigma$. On the other hand, we can consider to use a larger j, violating the assumed condition $(m/j)^2 \leq h$ in order to reduce the verification time. However, the search time will not be reduced and therefore the time bound cannot decrease.

5 The Grays Model

In the Grays model a match requires that the color of text cell must be between the minimum and maximum pattern colors in a neighborhood of the pattern cell that corresponds to the text cell. In this case, we do not enter into a single branch of the sistring trie, but for each pattern cell we follow all branches where color is between the minimum and maximum neighbor of that pattern cell. The number of characters qualifying for the next pattern character is a random variable that we call Δ, where $1 \leq \Delta \leq \sigma$.

Since there are now $O(\Delta^\ell)$ possible strings that match the pattern prefix of length ℓ, we touch

$$\sum_{\ell=1}^{h} \ell^{3/2} \Delta^\ell \ = \ O(h^{3/2} \Delta^h)$$

sistring trie nodes up to depth h, because all those qualifying strings exist up to that depth. From that depth on, there are on average $O(n^2/\sigma^\ell)$ sistrings in the text matching a given string of length ℓ. Therefore, the work in the deeper part of the sistring trie is

$$\sum_{\ell > h} \ell^{3/2} \ \frac{n^2}{\sigma^\ell} \ \Delta^\ell \ = \ O\left(h^{3/2} \ \frac{n^2}{\sigma^h} \ \Delta^h \right) \ = \ O(h^{3/2} \Delta^h)$$

since the first term of the summation dominates the rest. Therefore, the total complexity is

$$O(h^{3/2} \Delta^h) \ = \ O\left((\log_\sigma n)^{3/2} n^{2 \log_\sigma \Delta} \right) \ = \ O\left((\log_\sigma n)^{3/2} n^{2(1 - \log_\sigma 5/4)} \right).$$

Here the last step is based on that $\overline{\Delta}$, the average value of Δ, equals $(4/5)\sigma$ as the difference between the maximum and minimum of 9 values uniformly distributed over σ On the other hand, the cost function is concave in terms of Δ, and hence $\overline{f(\Delta)} \leq f(\overline{\Delta})$. In practice $\overline{\Delta}$ is much less than $(4/5)\sigma$, see [5].

6 The Accumulated Model

Even more powerful model is the Accumulated model, which provides a Hamming-like matching capability for gray-level images. Here, the sum of the absolute differences between text colors and the color of the corresponding pattern cell must not exceed k.

As for the Hamming model, we have to enter, for each relevant rotation, into all the branches of the sistring trie until we obtain an accumulated difference larger than k. We present first a standard approach and then a pattern partitioning technique.

6.1 Standard Searching

We enter into all the branches of the sistring trie until we can report a match or the sum of the differences exceeds k. As we show in Appendix B, the number of strings matching a given string of length ℓ under this model is at most $2^\ell \binom{k+\ell}{k}$. Since up to length h all them exist, we traverse

$$\sum_{\ell=1}^{h} \ell^{3/2} 2^\ell \binom{k+\ell}{k} \quad = \quad O\left(h^{3/2} 2^h \binom{k+h}{k}\right)$$

nodes in the trie. For the deeper parts of the trie there are $O(n^2/\sigma^\ell)$ strings matching a given one on average, so the rest of the search takes

$$\sum_{\ell>h} \ell^{3/2} \ \frac{n^2}{\sigma^\ell} \ 2^\ell \binom{k+\ell}{k} \quad = \quad \sum_{\ell>h} \ell^{3/2} \ n^2 \ \frac{2^\ell}{\sigma^\ell} \binom{k+\ell}{k}$$

In Appendix B we show that $(2/\sigma)^\ell \binom{k+\ell}{k}$ is exponentially decreasing with ℓ for $k/\ell < \sigma/(2e) - 1$, otherwise it is $\Omega(1/\sqrt{\ell})$. Therefore, we define

$$k_A^* \quad = \quad \frac{k}{\sigma/(2e) - 1}$$

and if $h \leq k_A^*$ the summation is at least $O(hn^2)$ and therefore not sublinear. If, on the other hand, $h > k_A^*$, then the first term of the summation dominates the rest, for a total search cost of

$$O\left(h^{3/2} 2^h \binom{k+h}{k}\right) \quad = \quad O\left((\log_\sigma n)^{3/2} n^{2\log_\sigma 2} \left(1 + \frac{2\log_\sigma n}{k}\right)^k\right)$$

which is sublinear in n for $\sigma > 2$. On the other hand, $\sigma = 2$ means a bilevel image, where the Hamming model is the adequate choice. Hence we obtain sublinear complexity (albeit exponential on k) for $k_A^* < 2\log_\sigma n$.

6.2 Pattern Partitioning

As for the Hamming model, we can partition the pattern to j^2 subpatterns that are searched exhaustively in the sistring trie. Again considering $(m/j)^2 \leq h$ we have a total search cost of

$$j^2 m^3 \left(U \ + \ Uk \frac{n^2}{\sigma^{(m/j)^2}} \right)$$

where this time

$$U = 2^{(m/j)^2} \binom{(m/j)^2 + k/j^2}{k/j^2}$$

After distributing the initial factor of the cost formula, we see that the first term decreases and the second term increases as a function of j. The optimum is found when both terms meet, which is again $j \ = \ m/\sqrt{2 \log_\sigma n}$ which is consistent with our condition $(m/j)^2 \leq h$. In fact, the second term is decreasing only for $\alpha < \sigma/(2e) - 1$, otherwise the optimum is $j = 1$, i.e. no pattern partitioning.

For this optimal j, the overall complexity is

$$O\left(\frac{m^5}{\log_\sigma n} \ n^{2(\log_\sigma 2 + H^A_\sigma(\alpha))} \right)$$

where we have defined $H^A_\sigma(\alpha) = -\alpha \log_\sigma(\alpha) + (1+\alpha)\log_\sigma(1+\alpha)$.

This complexity is sublinear as long as $\alpha < \sigma/(2e) - 1$. Again, we can consider to use a larger j value but the complexity does not improve.

7 An Alternative Matching Model

We have considered up to now that text centers match the value of the pattern cells they lie in. This has been done for technical convenience, although an equally reasonable alternative model is that the pattern cells must match the text color where their centers lie in the text.

Except for the Grays model, all the algorithms considered can be adapted to this case. The algorithms are more complex in practice now, because there may be more than one pattern center lying at the same text cell, and even no pattern center at all. This means that in some branches of the sistring trie we may have more than one condition to fit (which may be incompatible and then the branch can be abandoned under some models) and there may be no condition at all, in which case we have to follow all the branches at that level of the trie.

On average, however, we still have $\Theta(\ell)$ conditions when entering in the sistring trie with a pattern string of length ℓ, and therefore all the time bounds remain the same. However, in the Exact matching model, we can do better using the pattern centers.

We consider now indexing the rotated versions of the text sistrings, instead of considering the rotated versions of the pattern at search time. Hence, the pattern is simply searched with no rotations. Imagine that we index all the rotations of

the text up to depth H. This means that there will be $O(n^2H^{3/2})$ sistrings, and the sizes of the sistring trie and array will grow accordingly.

The benefit comes at search time: in the first part of the search we do not need to consider rotations of the pattern, since all the rotated ways to read the text are already indexed. Since we index $O(n^2H^{3/2})$ strings now, all the different sistrings will exist until depth $h' = \log_\sigma(n^2H^{3/2}) = 2\log_\sigma n + 3/2\log_\sigma H$. We first assume that $H \geq h'$. This means that until depth H we pay $O(H)$. After that depth all the surviving rotations are considered. Since $H \geq h'$, they all yield different strings, and the summation, as in Section 3, yields $O(n^2H^{3/2}/\sigma^H)$. Therefore the total search time is

$$O\left(H + \frac{n^2H^{3/2}}{\sigma^H}\right)$$

which is optimized for $H = 2\log_\sigma n + (1/2)\log_\sigma H$. Since this is smaller than h' we take the minimal $H = h'$. For instance $H = x\log_\sigma n$ works for any $x > 2$.

This makes the total search time $O(\log_\sigma n)$ on average. The space complexity becomes now $O(n^2(\log_\sigma n)^{3/2})$. Trying to use $H < h'$ worsens the complexity.

The matching model has changed, however. In the normal index the text sistrings are indexed once at a fixed rotation (zero). When a given pattern rotation is tried, the pattern is read in rotated form, in an order driven by the text centers. Now the text sistrings are read in all the rotated forms, and the pattern will be read once. The way to index a text sistring in rotated form is to assume that a rotated pattern is superimposed onto it and read the text cells where the pattern cells, read in order, fall. This effectively corresponds to the model we are considering in this section.

8 Experimental Results (Preliminary)

We have implemented the algorithms for Exact and Accumulated models, without the pattern partitioning technique. For the index structure, we used sistring array. The array based implementation is much more space efficient, but the search cost is also higher (both asymptotically and by the constant factors).

The implementation is in C, compiled using `gcc 2.95.2` on Linux 2.0.38, running in 700MHz PentiumIII machine. The implementation is not very optimized, and much of the time is spent in trigonometry; for computing the next angle to try, and for computing the coordinates of the cells for the given angle. Our test text was an image of size 768×768 cells, with 35 colors (gray levels), and a pattern of size 41×41 was extracted from it.

Table 2 shows some timings for the search. The difference between the times of the Exact model and the Accumulated model with $k = 0$ reveals the more complex implementation of the Accumulated model. Our results show that the algorithms can be implemented, and although preliminary versions, they run reasonably fast. For comparison, our optimized $O(n^2(k/\sigma)^{3/2})$ expected time on–line algorithm [4] spends 0.81 seconds for $k = 0$, 1.67 seconds for $k = 8$, 3.62 seconds for $k = 192$, and 3.85 seconds for $k = 256$. With the pattern partitioning, the algorithms would be much faster for large k.

k	Exact	0	1	2	4	8	16	32	64	96	128	192	256
time	0.0055	0.0086	0.0088	0.0089	0.0090	0.0097	0.0110	0.0165	0.0567	0.1720	0.3930	2.0390	6.3910

Table 2. Experimental results for the Exact and Accumulated models. The times are given in seconds.

9 Conclusions and Future Work

We have proposed a sistring tree index to search two dimensional patterns in two dimensional texts allowing rotations. We have considered different matching models and obtained average time bounds that are sublinear for most reasonable cases.

It is possible to extend the model by removing the center–to–center assumption [4]. In this case the number of patterns grows as high as $O(m^7)$ and therefore there are $O(\ell^{7/2})$ sistrings to search at depth ℓ. The search time for the Exact model becomes $O(\log_\sigma n)^{9/2}$. By indexing all the rotations and center displacements we get $O(\log_\sigma n)$ time again, but at a space cost of $O(n^2(\log_\sigma n)^{7/2})$.

It is also possible to extend the methods to three dimensions [6]. With the center–to–center assumption we have $O(m^{11})$ rotations. This means $O(\ell^{11/3})$ sistrings at depth ℓ. Therefore, at $O(n^3)$ space the total search time becomes $O((\log_\sigma n)^{14/3})$ for exact searching. If we index all the rotations up to $H = x \log_\sigma n$ with $x > 3$ we will have a space requirement of $O(n^3(\log_\sigma n)^{11/3})$ and a search cost of $O(\log_\sigma n)$. For the Grays model we have $O((\log_\sigma n)^{11/3} n^{3(1-\log_\sigma (28/27))})$ time. All the other techniques can be extended as well.

References

1. A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. In N. Alon, editor, *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 59–68, Victoria, B.C., Canada, May 1992. ACM Press.
2. A. Amir. Multidimensional pattern matching: A survey. Technical Report GIT-CC-92/29, Georgia Institute of Technology, College of Computing, 1992.
3. K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM'98)*, LNCS 1448, pages 118–125, 1998.
4. K. Fredriksson and E. Ukkonen. Algorithms for 2–d hamming distance under rotations. Manuscript, 1999.
5. K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. *Pattern Recognition Letters*, 20(11–13):1249–1258, 1999.
6. K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. Submitted, 2000.
7. Z. Galil and K. Park. Truly alphabet-independent two-dimensional pattern matching. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 247–257, Pittsburgh, PN, October 1992. IEEE Computer Society Press.

8. R. Giancarlo. A generalization of suffix trees to square matrices, with applications. *SIAM J. on Computing*, 24:520–562, 1995.
9. R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Information and Computation*, 130:151–182, 1996.
10. G. H. Gonnet. Efficient searching of text and pictures. Report OED-88-02, University of Waterloo, 1988.
11. J. Kärkkäinen and E. Ukkonen. Two and higher dimensional pattern matching in optimal expected time. In Daniel D. Sleator, editor, *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 715–723, Arlington, VA, January 1994. ACM Press.
12. G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12(4/5):375–408, October 1994.
13. G. Navarro and R. Baeza-Yates. Fast multi-dimensional approximate string matching. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS, pages 243–257, 1999.
14. T. Takaoka. Approximate pattern matching with grey scale values. In Michael E. Houle and Peter Eades, editors, *Proceedings of Conference on Computing: The Australian Theory Symposium*, pages 196–203, Townsville, January 29–30 1996. Australian Computer Science Communications.
15. J. Tarhio. A sublinear algorithm for two-dimensional string matching. *PRL: Pattern Recognition Letters*, 17, 1996.
16. J. Wood. Invariant pattern recognition: a review. *Pattern Recognition*, 29(1):1–17, 1996.

A Probability of Matching under the Hamming Model

We need to determine which is the probability of the search being active at a given node of depth ℓ in the sistring trie under the Hamming model. We are therefore interested in the probability of a pattern prefix of length ℓ matching a text substring of length ℓ. For this to hold, at least $\ell - k$ text characters text must match the pattern. Hence, the probability of matching is upper bounded by

$$\frac{1}{\sigma^{\ell-k}}\binom{\ell}{k}$$

where the combinatorial counts all the possible locations for the matching characters.

In the analysis that follows, we call $\beta = k/\ell$ and take it as a constant (which is our case of interest, as seen later). We will prove that, after some length ℓ, the matching probability is $O(\gamma(\beta)^\ell)$, for some $\gamma(\beta) < 1$. By using Stirling's approximation $x! = (x/e)^x\sqrt{2\pi x}(1 + O(1/x))$ over the matching probability we have

$$\frac{1}{\sigma^{\ell-k}}\left(\frac{\ell^\ell\sqrt{2\pi\ell}}{k^k(\ell-k)^{\ell-k}\sqrt{2\pi k}\sqrt{2\pi(\ell-k)}}\right)\left(1+O\left(\frac{1}{\ell}\right)\right)$$

which is

$$\left(\frac{1}{\sigma^{1-\beta}\beta^\beta(1-\beta)^{1-\beta}}\right)^\ell \ \ell^{-1/2}\ \left(\frac{1}{\sqrt{2\pi\beta(1-\beta)}}+O\left(\frac{1}{\ell}\right)\right)$$

This formula is of the form $\gamma(\beta)^\ell\ O(1/\sqrt{\ell})$, where we define

$$\gamma(x) \quad = \quad \frac{1}{\sigma^{1-x}x^x(1-x)^{1-x}}$$

Therefore the probability is exponentially decreasing with ℓ if and only if $\gamma(\beta) < 1$, that is,

$$\sigma \quad > \quad \left(\frac{1}{\beta^\beta(1-\beta)^{1-\beta}}\right)^{\frac{1}{1-\beta}} \quad = \quad \frac{1}{\beta^{\frac{\beta}{1-\beta}}(1-\beta)}$$

It is easy to show analytically that $e^{-1} \leq \beta^{\frac{\beta}{1-\beta}} \leq 1$ if $0 \leq \beta \leq 1$, so it suffices that $\sigma > e/(1-\beta)$, or equivalently, $\beta < 1 - e/\sigma$ is a sufficient condition for the probability to be exponentially decreasing with ℓ.

Hence, the result is that the matching probability is very high for $\beta = k/\ell \geq 1 - e/\sigma$, and that otherwise it is $O(\gamma(\beta)^\ell/\sqrt{\ell})$, where $\gamma(\beta) < 1$.

B Probability of Matching under the Accumulated Model

We need to determine what is the probability of the search being active at a given node of depth ℓ in the sistring trie under the Accumulated model. We are therefore interested in the probability of two random strings of length ℓ matching with at most k errors. Our model is as follows: we consider the sequence of ℓ absolute differences between both strings $\delta_1 \ldots \delta_\ell$. The matching condition states that $\sum_{i=1}^{\ell} \delta_i \leq k$.

The number of different sequences of differences satisfying this is $\binom{k+\ell}{\ell}$, what can be seen as the number of ways to insert ℓ divisions into a sequence of k elements. The ℓ divisions divide the sequence in $\ell + 1$ zones. The sizes of the first ℓ zones are the δ_i values and the last allows the sum to be $\leq k$ instead of exactly k. Note that we are pessimistically forgetting about the fact that indeed $\delta_i \leq \sigma$.

Finally, each difference δ_i can be obtained in two ways: $P_i + \delta_i$ and $P_i - \delta_i$ (we again pessimistically count twice the case $\delta_i = 0$). Therefore, the total matching probability is upper bounded by

$$\frac{2^\ell}{\sigma^\ell}\binom{\ell+k}{k}$$

In the analysis that follows, we call $\beta = k/\ell$ and take it as a constant (which is our case of interest, as seen later). We will prove that, after some length ℓ, the matching probability is $O(\gamma(\beta)^\ell)$, for some $\gamma(\beta) < 1$. By using Stirling's approximation $x! = (x/e)^x\sqrt{2\pi x}(1 + O(1/x))$ over the matching probability we have

$$\frac{2^\ell}{\sigma^\ell}\left(\frac{(k+\ell)^{k+\ell}\sqrt{2\pi(k+\ell)}}{k^k\ell^\ell\sqrt{2\pi k}\sqrt{2\pi\ell}}\right)\left(1 + O\left(\frac{1}{\ell}\right)\right)$$

which is

$$\left(\frac{2(1+\beta)^{1+\beta}}{\sigma\beta^{\beta}}\right)^{\ell} \ell^{-1/2} \left(\sqrt{\frac{1+\beta}{2\pi\beta}} + O\left(\frac{1}{\ell}\right)\right)$$

This formula is of the form $\gamma(\beta)^{\ell}\, O(1/\sqrt{\ell})$, where we define

$$\gamma(x) \quad = \quad \frac{2(1+x)^{1+x}}{\sigma x^{x}}$$

Therefore the probability is exponentially decreasing with ℓ if and only if $\gamma(\beta) < 1$, that is,

$$\frac{2(1+\beta)}{\sigma}\left(1+\frac{1}{\beta}\right)^{\beta} \quad < \quad 1$$

It can be easily seen analytically that $(1+1/\beta)^{\beta} \leq e$, so $\beta < \sigma/(2e) - 1$ is a sufficient condition for the probability to be exponentially decreasing with ℓ.

Hence, the result is that the matching probability is very high for $\beta = k/\ell \geq \sigma/(2e) - 1$, and that otherwise it is $O(\gamma(\beta)^{\ell}/\sqrt{\ell})$, where $\gamma(\beta) < 1$.

Parallel Edge Coloring of a Tree on a Mesh Connected Computer*

Chang-Sung Jeong, Sung-Up Cho, Sun-Chul Whang, and Mi-Young Choi

Department of Electronics Engineering, Korea University
Anamdong 5-ka, Sungbuk-ku, Seoul 136-701, Korea
Tel: 82-2-3290-3229, Fax: 921-0544, E-mail: csjeong@charlie.korea.ac.kr

Abstract. An edge coloring is an assignment of colors to the edges of a graph so that no two edges with a common vertex have the same color. We show that, given an undirected tree T with n vertices, a minimum edge coloring of T can be determined in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ mesh-connected computer(MCC) by a novel technique which decomposes the tree into disjoint chains and then assigns the edge colors in each chain properly. The time complexity is optimal on MCC within constant factor.

1 Introduction

A mesh-connected computer(MCC) consists of n identical processing elements(PE's) arranged on a $\sqrt{n} \times \sqrt{n}$ array, where each PE is connected to its four neighbors [1]. (See Fig. 1.) We assume that MCC functions in SIMD mode, where all the PE's are synchronized under one control unit. MCC have been used as a model of parallel computation for problems in diverse areas including sorting, graph theoretic problems, computational geometry, image processing [2, 3,4,5,6,7]. In this paper, we consider the problem of edge coloring in graph theory on MCC.

An edge coloring is an assignment of colors to the edges of a graph $G = (V, E)$ so that no two edges with a common vertex have the same color. A minimum edge coloring is an edge coloring which uses a minimum number of colors. Many researchers have worked on the design of sequential algorithms for finding a minimum edge coloring of a bipartite graph [10,11,12,13,14]. The best known sequential algorithm for edge coloring of a bipartite graph is due to Cole and Hopcroft [10] and runs in $O(|E|log|V|)$ time. For a minimum edge coloring of a tree T with n vertices, Mitchell and Hedetniemi presented an $O(n)$ sequential algorithm [14].

Using the idea presented in [11,12], Lev, Pippenger and Valiant implemented an $O(log^3 n)$ parallel algorithm for a minimum edge coloring of a bipartite graph on EREW(Exclusive Read and Exclusive Write) computation model [13]. However, their algorithm cannot be directly implemented for tree on MCC in optimal time. As far as we know, no papers have proposed an optimal parallel

* This work has been supported by KOSEF and Brain Korea 21 projects

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 76–83, 2000.
© Springer-Verlag Berlin Heidelberg 2000

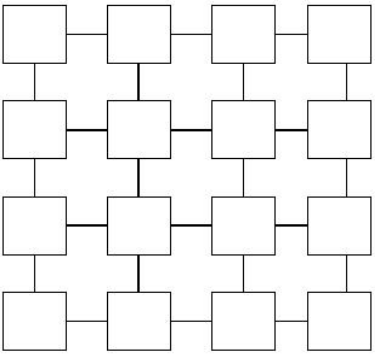


Fig. 1. 4×4 Mesh-connected computer

algorithm for the minimum edge coloring of a tree so far. In this paper, we consider the problem of minimum edge coloring of a tree with n vertices and present an $O(\sqrt{n})$ parallel algorithm for solving the problem on a $\sqrt{n} \times \sqrt{n}$ MCC, which is optimal within constant time factor. We shall show that this can be achieved by a novel technique which decomposes the tree into disjoint chains and then assigns the edge colors in each chain properly.

In section 2, we describe some notations and definitions, and in section 3 basic operations used in the design of the parallel algorithm. In section 4, we give an $O(\sqrt{n})$ parallel edge coloring algorithm for a tree $T(V, E)$, $|V| = n$, on a $\sqrt{n} \times \sqrt{n}$ MCC and in section 5 we give a conclusion.

2 Notations and Definitions

Let $T(V, E)$ be a rooted directed tree, where there is a unique path from the root to each vertex in V. The vertices in V are labeled from 1 to $|V|$. If (i, j) is in edge set E, then i is the *parent* of j, denoted by $p(j)$, and j is a *child* of i. Note that for each vertex i, its parent $p(i)$ is unique, whereas it may have more than one child. The *depth* of a vertex v in V is the length(i.e., number of edges) of the path from the root to v. The *degree* of a vertex v in T is the number of the edges incident upon v in T. Let d be the maximum degree of a vertex in T. The minimum number of colors required in an edge coloring of a graph G is called the *edge chromatic number* of G. It is well known that the edge chromatic number of a bipartite graph G is equal to the maximum degree of G [9]. Since a tree is a bipartite graph, the edge chromatic number of T becomes d. Therefore, the minimum edge coloring of T using colors 1 through d partitions E into $E_1, E_2, .., E_d$ such that all the edges in E_i are assigned color i and no two edges in E_i are adjacent to each other. (See Fig. 2.)

The *order number* $s(i)$, for each edge $(p(i), i)$ in T, is defined as follows: We order each set S of edges with the same parent by sorting the edges $(p(i), i)$ in S in ascending order of i. For each edge $(p(i), i)$, we define $s(i)$ to be the number of

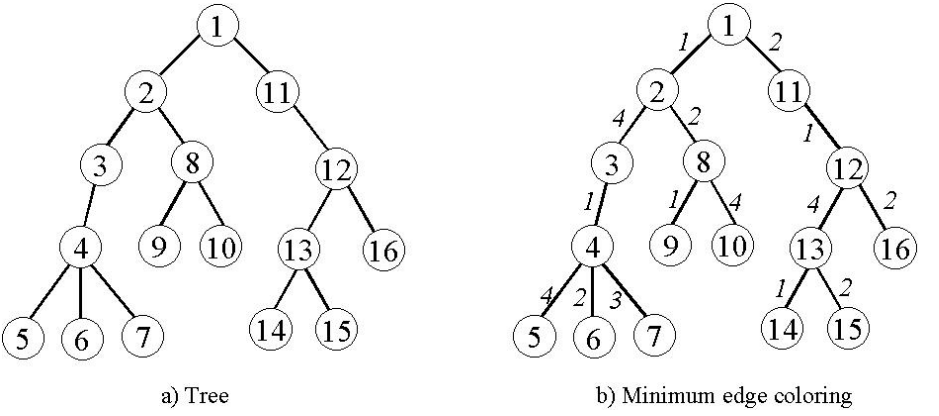


Fig. 2. Edge coloring of a tree with maximum degree 4

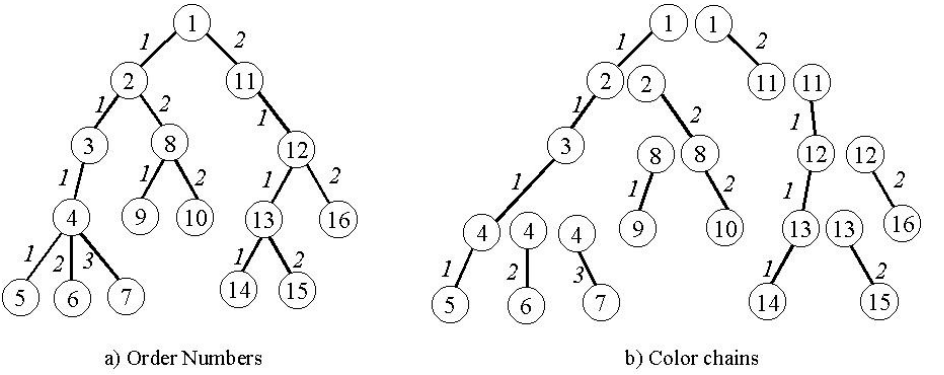


Fig. 3. Order numbers and color chains

the edges in S previous to it including itself. (See Fig. 3-a.) Suppose each edge in T is colored with its order number. A path in T is called a *color chain* if it is a maximal path consisting solely of edges with the same colors. (See Fig. 3-b.) A color chain consisting of edges with color k is called *k-chain*.

A *linear chain* is a directed path which consists of edges each of the form $(i, succ(i))$, where vertex $succ(i)$ is the immediate successor of vertex i. (See Fig. 4.) Suppose that each edge is associated with its weight. Then, the *rank* of an edge in the linear chain is the sum of the weights of its preceding edges and itself. Note that each color chain is a linear chain if i, for each edge $(p(i), i)$, is regarded as the immediate successor of $p(i)$. The *root edge* in the color chain(or linear chain) is the one with no predecessors.

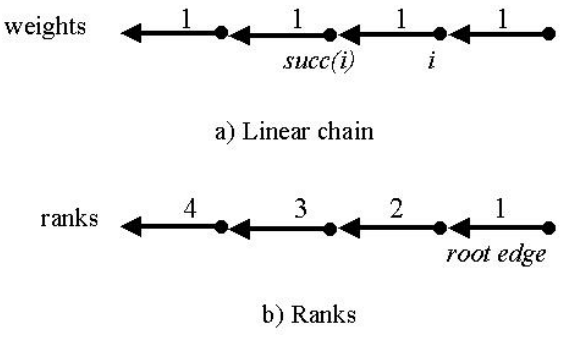


Fig. 4. Rank in a linear chain

3 Basic Operations

MCC consists of n PE's indexed from 0 to $n-1$, each having a local memory with a constant number of registers. We shall use the following basic operations which take $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ MCC [2,3,5,8]. We assume that input data of size n are distributed on a $\sqrt{n} \times \sqrt{n}$ MCC, one element per PE.

(1). Sorting: The elements are rearranged in nondecreasing or nonincreasing order of a specified key.

(2). Selected Broadcasting: Suppose a set S of n elements is partitioned into subsets $\{S_1, S_2, .., S_m\}$. By *selected broadcasting* operation, the first element of each S_i is distributed to all the other PE's containing the elements in S_i.

(3). RAR(Random Access Read): In a RAR, each PE specifies the key of data it wishes to receive, or it specifies a null key, in which case it receives nothing. Each PE reads a data from the PE which contains its specified key. Several PE's can request the same key and read the same data.

(4). RAW(Random Access Write): In a RAW, each PE specifies the key of data, and send its data to the PE which contains its specified key. If two or more PE's send data with the same specified key, then the PE with that key will receive the data with the minimium data or other value obtained by applying some commutative, associative binary operation to all the data sent to it.

(5). Chain-Rank: A linear chain consists of directed edges each of which is associated with its weight. The rank of each edge is defined as the sum of the weights of its predecessors in the chain. The first edge in the linear chain is called a *root* edge. Given a collection of linear chains with n edges, *chain-rank* operation finds, for each edge, its *rank* in the linear chain containing it. Figure 4 shows an example of linear chain and ranks of its edges each with weight 1.

4 Parallel Algorithm

In this section we shall describe a parallel algorithm for finding a minimum edge coloring for a tree T. The basic idea of our algorithm can be described briefly as follows: First, color every edge of T by its order number. Note that no more than two edges adjacent to a vertex in T have the same color, and color d have not been used at all, since the maximum value of order numbers is $d-1$. Using those facts, we can obtain a minimum edge coloring by decomposing T into disjoint color chains and then changing the colors of every other edge in each color chain into d. In the following we shall describe the detailed implementation of the parallel algorithm.

Parallel Algorithm Edge-Coloring(EC)

Input: Each PE with index i on a $\sqrt{n} \times \sqrt{n}$ MCC contains an edge $(p(i), i)$ and a vertex i of a directed tree $T(V, E)$.

Output: A minimum edge coloring C such that $C(p(i), i)$ is m if the edge $(p(i), i)$ is assigned color m.

1): Find a maximum degree d for T. This can be done as follows: Send, for each PE containing an edge $(p(i), i)$, 1 to the PE containing a vertex $p(i)$ by RAW. During RAW, several 1's may have the same destination PE, and the PE containing a vertex j can receive the value obtained by summing all the 1's sent to that PE. Therefore, each PE containing a vertex j stores the number of its children after executing RAW, and hence we can compute the degree of j by adding one to it. Then, we can find easily the maximum degree d by computing the largest value of degrees of all the vertices.

2): Compute, for each edge $(p(i), i)$, its order number $s(i)$.

Implementation: First, sort every edge $(p(i), i)$ lexicographycally according to $(p(i), i)$ in nondecreasing order. During sort the smallest edge e in each set S of edges with the same parent is located and index I of PE containing e is sent to all the PE's storing the other edges in S by *selected broadcasting*. Then, $s(i)$ can be obtained by subtracting $(I-1)$ from the PE index containing $(p(i), i)$.

3): Assign, for each edge $(p(i), i)$, color $s(i)$ to $C(p(i), i)$. (See Fig. 3-a.) The colors assigned at this step range from 1 to $d-1$, since the maximum value of $s(i)$ is $d-1$.

4): Decompose T into disjoint k-chains, $1 \leq k \leq d-1$. (See Fig. 3-b.)

Implementation: Step 4) can be done by seperating the root edge in each color chain from the other color chains. This can be implemented as follows: Find, for each edge $(p(i), i)$, its parent edge $(j, p(i))$ by RAR, where j is a parent of $p(i)$. Then, check whether the color of $(p(i), i)$ is different from that of $(j, p(i))$ or not. If it is, seperate $(p(i), i)$ from the other color chains adjacent to it by replacing $(p(i), i)$ with $(p(i)_{s(i)}, i)$, since $(p(i), i)$ becomes a root edge of the color chain containing it; otherwise $(p(i), i)$ cannnot be a root edge of

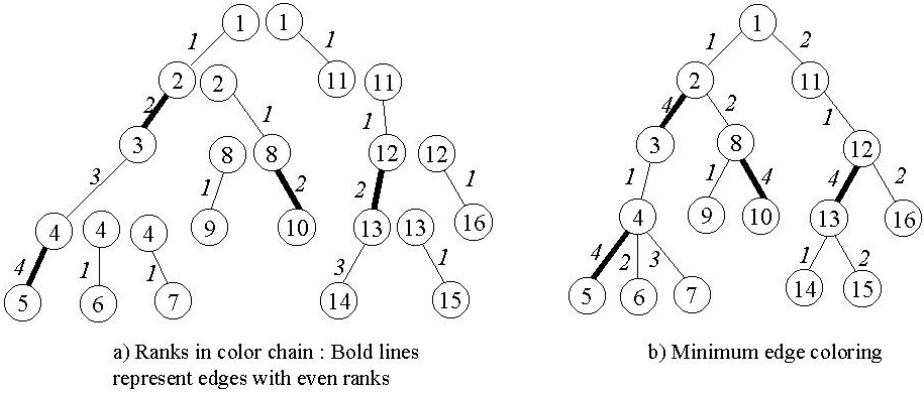


Fig. 5. Example of the parallel algorithm EC

the color chain to which it belongs, and hence we don't have to separate it from the other color chains.

5): Find, for each edge $(p(i), i)$, its rank in the color chain containing it after assigning a weight 1 to every edge. (See Fig. 5-a.)

Comment: We can easily see that at most two edges with a common vertex may have the same color after executing step 3). In order to fix the conflicting color assignments, the colors of all the edges with even ranks are replaced with color d in the following step.

6): Change, for each edge $(p(i), i)$ with the even rank, its color $C(p(i), i)$ into d. (See Fig. 5-b.)

End Algorithm.

Lemma 1: The parallel algorithm EC can be executed in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ MCC.
Proof: Step 1) and 2) take $O(\sqrt{n})$ time for RAW, sorting, and *selected broadcasting*. Step 3) can be done in O(1) time and step 4) in $O(\sqrt{n})$ time for RAR. Step 5) can be carried out in $O(\sqrt{n})$ time by *chain-rank* operation. Step 6) takes O(1) time. Therefore, the overall time complexity is $O(\sqrt{n})$ time.

Lemma 2: The edge coloring C in the parallel algorithm EC is a minimum edge coloring for T.
Proof: Clearly, the colors used in the parallel algorithm EC range from 1 to d. Since d is a edge chromatic number of T, all we have to prove is that no two edges with a common vertex have the same color. Consider a vertex i with parent $p(i)$ and m children $j_1, j_2, j_3, .., j_m$ ordered from left to right so that the order number of (i, j_k) is k, $1 \leq k \leq m$. (See Fig. 6-a.) Let A be a set of edges $(i, j_k), 1 \leq k \leq m$. After step 3) each edge (i, j_k) is assigned color k in the range

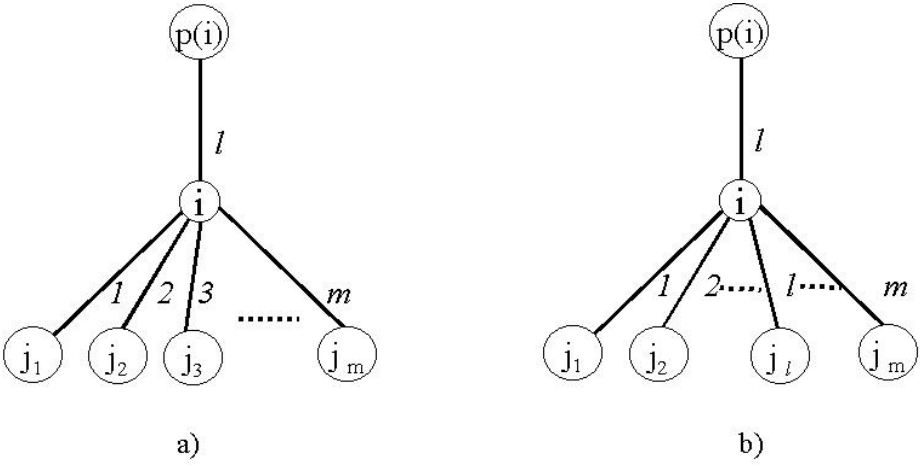


Fig. 6. Illustration of Lemma 2

from 1 to $d-1$. Suppose that edge $(p(i), i)$ is colored ℓ. Then, we can cosider the following two cases:

Case 1): $\ell > m$: Since all the edges in A are assigned colors all different from ℓ, each edge in A becomes a root edge in the color chain containing it and has its rank value 1. Therefore, the color of each edge in A is not changed at step 6), and hence no two edges adjacent to i have the same color.
Case 2): $\ell \leq m$: Let (i, j_ℓ) denote an edge with the same color ℓ as $(p(i), i)$. (See Fig. 6-b.) Since both of $(p(i), i)$ and (i, j_ℓ) belong to the same ℓ-chain, one of them changes its color ℓ to d at step 6). All the edges in A other than (i, j_ℓ) have different colors from ℓ, and their colors are not changed due to the fact that each of them is a root edge in the color chain containing it. Therefore, no two edges adjacent to i have the same color.

An undirected tree with n vertices can be converted into the directed one in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ MCC [5]. Therefore, the following theorem follows directly from lemma 1 and 2.

Theorem 1: Given an undirected tree T with n vertices, its minimum edge coloring can be computed in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ MCC.

5 Conclusion

In this paper, we have presented the parallel algorithm for finding the minimum edge coloring of a tree with n vertices in $O(\sqrt{n})$ time on a $\sqrt{n} \times \sqrt{n}$ MCC. The time complexity is optimal on MCC within constant factor. This was achieved by the smart technique which decomposes the tree into disjoint color chains and then changes the colors of each color chain properly. It remains open if one

can find an $O(\sqrt{n})$ algorithm for finding a maximum matching of a tree with n vertices on a $\sqrt{n} \times \sqrt{n}$ MCC. Also open are the problems for finding a minimum edge coloring and maximal matching for a general bipartite graph on a $\sqrt{n} \times \sqrt{n}$ MCC in $O(\sqrt{n})$ time. We expect that our parallel algorithm developed in this paper may provide a base for finding the optimal parallel algorithms for those open problems.

References

1. H. J. Siegel, A Model of SIMD Machines and a Comparison of Various Interconnection Networks, *IEEE Trans. Comput. 28* (1979) 907-917.
2. C. S. Jeong, and D. T. Lee, Parallel Geometric Algorithms on a Mesh-Connected Computer, *Algorithmica 5* (1990) 155-177.
3. C. D. Thompson, and H. T. Kung, Sorting on a Mesh–Connected Parallel Computer, *Comm. ACM 20* (1977) 263–271.
4. N. Folwell, S. Guha and I. Suzuki, A Practical Algorithm for Integer Sorting on a Mesh-Connected Computer, *Parallel Algorithms and Applicationsi 12* (1997) 265-278.
5. M. M. Atallah, and S. E. Hambrusch, Solving Tree Problems on a Mesh-Connected Processor Array, *Proc. 26th IEEE FOCS Conference* (1985) 222-231.
6. M. H. Kim, and C. S. Jeong, An Optimal Parallel Matching Algorithm for a Convex Bipartite Graph on a Mesh-Connected Computer, *Parallel Algorithms and Applications 5* (1995) 15-35.
7. B. Murthy, K. Bhuvaneswari, and C. S. Murthy, A New Algorithm based on Givens Rotations Solving Linear Equations on Fault-Tolerant Mesh-Connected Processors, *IEEE Trans. on Parallel and Distributed Systems 9* (1998) 825-832.
8. D. Nassimi, and S. Sahni, Data Broadcasting in SIMD Computers, *IEEE Trans. on Comput.* (1981) 101-106.
9. J. A. Bondy, and U. S. R. Murty, Graph Theory with Applications, *American Elsevier* (1977).
10. R. Cole, and J. Hopcroft, On Edge Coloring Bipartite Graphs, *SIAM J. Comput. 11* (1982) 540-546.
11. H. Gabow, Using Euler Partitions to Edge Color Bipartite Multigraphs, *Inter. J. Comp. and Inf. Scie. 5* (1976) 345-355.
12. H. Gabow, and O. Kariv, Algorithms for Edge Coloring Bipartite Multigraphs, *SIAM J. Comput. 11* (1982) 117-129.
13. G. Lev, N. Pippenger, and L. G. Valiant, A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE Trans. Comput. 30* (1981) 93-110.
14. S. Mitchell, and S. Hedetniemi, Linear Algorithms for Edge coloring Trees and Unicyclic Graphs, *Info. Proc. Lett 9* (1979) 110-112.

Parallel Approximation Algorithms for Maximum Weighted Matching in General Graphs

Ryuhei Uehara[1] and Zhi-Zhong Chen[2]

[1] Natural Science Faculty, Komazawa University
[2] Department of Mathematical Sciences, Tokyo Denki University.

Abstract. The problem of computing a matching of maximum weight in a given edge-weighted graph is not known to be P-hard or in RNC. This paper presents four parallel approximation algorithms for this problem. The first is an RNC-approximation scheme, i.e., an RNC algorithm that computes a matching of weight at least $1-\epsilon$ times the maximum for any given constant $\epsilon > 0$. The second one is an NC approximation algorithm achieving an approximation ratio of $\frac{1}{2+\epsilon}$ for any fixed $\epsilon > 0$. The third and fourth algorithms only need to know the total order of weights, so they are useful when the edge weights require a large amount of memories to represent. The third one is an NC approximation algorithm that finds a matching of weight at least $\frac{2}{3\Delta+2}$ times the maximum, where Δ is the maximum degree of the graph. The fourth one is an RNC algorithm that finds a matching of weight at least $\frac{1}{2\Delta+4}$ times the maximum on average, and runs in $O(\log \Delta)$ time, not depending on the size of the graph.

Keywords: Graph algorithm, maximum weighted matching, approximation algorithm, parallel algorithm.

1 Introduction

Throughout this paper, a graph means an edge-weighted graph, unless explicitly specified otherwise. A *matching* in a graph G is a set M of edges in G such that no two edges in M share an endpoint. The *weight* of a matching M is the total weight of the edges in M. The *maximum weighted matching* (MWM) problem is to find a matching of maximum weight of a given graph. The *maximum cardinality matching* (MCM) problem is the special case of the MWM problem where all edges of the input graph have the same weight.

For the MWM problem, Edmonds' algorithm [Edm65] has stood as one of the paradigms in the search of polynomial-time algorithms for integer programming problems (see also [Gal86]). Some sophisticated implementations of his algorithm have improved its time complexity: for example, Gabow [Gab90] has given an algorithm using $O(n(m + n \log n))$ time and $O(m)$ space.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 84–98, 2000.
© Springer-Verlag Berlin Heidelberg 2000

The parallel complexity of the MWM problem is still open; Galil asks if the MWM problem is P-complete [Gal86]. The best known lower bound is the CC-hardness result pointed out in [GHR95] (see [MS92] for the definition of the class CC). On the other hand, the best known upper bound is the RNC algorithm of Karp, Upfal, and Wigderson [KUW86] for the special case where the weights on the edges of the input graph G are bounded from above by a fixed polynomial in the number of vertices in G. We call this case the *polynomial-weight case* of the MWM problem.

From the practical point of view, Edmonds' algorithm is too slow. Faster algorithms are required, and heuristic algorithms and approximation algorithms have been widely investigated. A survey of heuristic algorithms can be found in [Avi83], and some approximation algorithms can be found in [Ven87]. In particular, a $\frac{1}{2}$-approximation algorithm can be obtained by a greedy strategy that picks up the heaviest edge e and deletes e and its incident edges, and repeats this until the given graph becomes empty. Recently, Preis [Pre99] gave a linear-time implementation of this greedy algorithm.

For the MCM problem, several NC approximation algorithms are known (see [KR98] for comprehensive reference). In particular, Fischer *et al.* [FGHP93] showed an NC algorithm that computes a matching with cardinality at least $1 - \epsilon$ times the maximum for any fixed constant $\epsilon > 0$. This NC approximation algorithm is based on essential properties of the MCM problem that do not belong to the MWM problem.

Our first result is an RNC approximation scheme for the MWM problem, i.e., an RNC algorithm which computes a matching of weight at least $1 - \epsilon$ times the maximum for any fixed constant $\epsilon > 0$. This scheme uses the RNC algorithm for the polynomial-weight case of the MWM problem due to Karp *et al.* as a subroutine.

Our second result is an NC approximation algorithm for the MWM problem. This algorithm can be viewed as a parallelized version of the greedy strategy. For any fixed constant $\epsilon > 0$, this algorithm computes a matching of weight at least $\frac{1}{2+\epsilon}$ times the maximum. The work done by the algorithm is optimal up to a polylogarithmic factor.

In the results above, the size of the problem depends not only on the number of edges and vertices, but also on space to represent the edge weights. Thus the algorithms are not useful when the edge weights require a large amount of memories to represent. We next consider the algorithms that only use the total order of the weights, and they do not need to store each weight itself.

Our third result is an NC approximation algorithm for the MWM problem. This algorithm computes a matching of weight at least $\frac{2}{3\Delta+2}$ times the maximum weight, where Δ is the maximum degree of given graph. It runs in $O(\log n)$ time using n processors.

Our fourth result is an RNC approximation algorithm for the MWM problem. This algorithm computes a matching whose expected weight is at least $\frac{1}{2\Delta+4}$ times the maximum weight. It runs in $O(\log \Delta)$ time using n processors. Remark

that the time complexity does not depend on the size of the graph; it can be performed efficiently in parallel even on a large scale distributed system.

The rest of the paper is organized as follows. In section 2, we give basic definitions of the problem. The first and second algorithms are discussed in section 3. The third and fourth algorithms for graphs with heavy weights are stated in section 4.

2 Preliminaries

We will deal only with graphs without loops or multiple edges. Let $G = (V, E)$ be a graph. For each edge e of G, $w_G(e)$ denotes the weight of e in G. We assume that each weight is positive, and it is different from other weights. For a subset F of edges of G, $w_G(F)$ denotes the total weight of the edges in F, i.e., $w_G(F) = \sum_{e \in F} w_G(e)$. A *maximal matching* of G is a matching of G that is not properly contained in another matching. A *maximum weighted matching* of G is a matching whose weight is maximized over all matchings of G. Note that a maximum weighted matching must be maximal. Let M be a matching and $\alpha = (e_1, e_2, \cdots, e_l)$ be a path of length l in a graph. We call α an *alternating path admitted by* M if edges on α are alternately in M, that is, either $\{e_1, e_3, \cdots\} \subseteq M$ or $\{e_2, e_4, \cdots\} \subseteq M$. We sometimes unify the alternating path and the set of edges in the matching.

The *neighborhood* of a vertex v in G, denoted by $N_G(v)$, is the set of vertices in G adjacent to v. The *degree* of a vertex v in G is $|N_G(v)|$, and denoted by $\deg_G(v)$. The *maximum degree* of a graph G is $\max_{v \in V} \deg_G(v)$, and denoted by Δ_G. Without loss of generality, we assume that $\Delta_G > 2$. The notations $w_G(e)$, $\deg_G(v)$, $N_G(v)$, and Δ_G are sometimes denoted by just $w(e)$, $\deg(v)$, $N(v)$, and Δ if G is understood. For $F \subseteq E$, $G[F]$ denotes the graph (V, F).

The model of parallel computation used here is the PRIORITY PRAM where the processors operate synchronously and share a common memory, both simultaneous reading and simultaneous writing to the same cell are allowed; in case of simultaneous writing, the processor with lowest index succeeds. An NC *algorithm* (respectively, RNC *algorithm*) is one that can be implemented to run in polylogarithmic time (respectively, expected polylogarithmic time) using a polynomial number of processors on a PRIORITY PRAM. More details about the model, NC algorithms and RNC algorithms can be found in [Joh90,KR90].

An algorithm $\mathcal{A}$ for the MWM problem *achieves a ratio* of ρ if for every graph G, the matching M found by $\mathcal{A}$ on input G has weight at least $\rho \cdot w_G(M^*)$, where M^* is a maximum weighted matching of G. An RNC *approximation scheme* for the MWM problem is a family $\{\mathcal{A}_i \ : \ i \geq 2\}$ of algorithms such that for any fixed integer $i \geq 2$, $\mathcal{A}_i$ is an RNC algorithm and achieves a ratio of $1 - \frac{1}{i}$.

3 Approximation Algorithms

3.1 Common Preprocess

Throughout the rest of this paper, let G be the input graph, M^* be a maximum weighted matching of G, and n and m be the number of vertices and edges in G, respectively. Let $w_{\max}$ be the maximum weight of an edge in G. Let k be a fixed positive integer and σ be a fixed positive real with $1 > \sigma > 0$; the actual values of k and σ will become clear later.

For each edge e of G, we define an integer $r(e)$ as follows:

- If $\frac{kn}{w_{\max}} \cdot w_G(e) \leq 1$, then let $r(e) = 0$.
- Otherwise, let $r(e)$ be the smallest integer i such that $\frac{kn}{w_{\max}} \cdot w_G(e) \leq (1+\sigma)^i$.

We call $r(e)$ the *rank* of e. Note that $r(e) \in \{0, 1, \cdots, \lceil \log_{1+\sigma} kn \rceil\}$.

Let E_i be the set of edges of G with rank i. Let $M_i^* = M^* \cap E_i$. The following lemma shows that the total weight of edges in M_0^* is significantly small.

Lemma 1. $\sum_{e \in M_0^*} w_G(e) \leq \frac{1}{2k} w_G(M^*)$.

Proof. For each edge $e \in E_0$, $w_G(e) \leq \frac{w_{\max}}{kn}$. Thus, $\sum_{e \in M_0^*} w(e) \leq \frac{n}{2} \frac{w_{\max}}{kn} \leq \frac{w_{\max}}{2k} \leq \frac{1}{2k} w(M^*)$. □

Let G' be the graph obtained from G by deleting all edges of rank 0 and modifying the weight of each rest edge e to be $(1+\sigma)^{r(e)}$.

3.2 RNC approximation scheme

In this section, we present the RNC approximation scheme for the MWM problem. It is based on the following result due to Karp *et al.*

Lemma 2. [KUW86] *There is an* RNC *algorithm for the polynomial-weight case of the MWM problem.*

Theorem 1. *There exists an* RNC *approximation scheme for the MWM problem.*

Proof. Let ℓ be an integer larger than 1. Suppose we want to compute a matching M of G with $w_G(M) \geq (1 - \frac{1}{\ell}) \cdot w_G(M^*)$. Then, we fix $\sigma = \frac{1}{\ell}$ and $k = \lceil \frac{\ell^2}{2} \rceil$, and construct G' as in Section 3.1. Note that the weight of each edge of G' is polynomial in n. So, we use the RNC algorithm in Lemma 2 to compute a maximum weighted matching M' of G'. Note that M' is also a matching of G. It remains to show that $w_G(M') \geq (1 - \frac{1}{\ell}) w_G(M^*)$.

For each $i \geq 1$, let $M_i' = M' \cap E_i$. Since M' is a maximum weighted matching of G' and $\cup_{i \geq 1} M_i^*$ is a matching of G', we have $\sum_{i \geq 1} \sum_{e \in M_i^*} w_{G'}(e) \leq w_{G'}(M') = \sum_{i \geq 1} \sum_{e \in M_i'} w_{G'}(e) = \sum_{i \geq 1} |M_i'| (1+\sigma)^i$.

On the other hand, since M^* is a maximum weighted matching of G, we have

$$w_G(M^*) \geq \sum_{i\geq 1}\sum_{e\in M'_i} w_G(e) > \sum_{i\geq 1} |M'_i| \frac{w_{\max}}{kn}(1+\sigma)^{i-1}$$

since $\frac{w_{\max}}{kn}(1+\sigma)^{i-1} < w_G(e)$ if $e \in M'_i$ with $i \geq 1$.

Thus, $w_G(M') = \sum_{i\geq 1}\sum_{e\in M'_i} w_G(e) \geq \frac{w_{\max}}{kn} \cdot \frac{\sum_{i\geq 1}|M'_i|(1+\sigma)^i}{(1+\sigma)}$
$\geq \frac{w_{\max}}{kn(1+\sigma)}\sum_{i\geq 1}\sum_{e\in M^*_i} w_{G'}(e) \geq \frac{w_{\max}}{kn(1+\sigma)}\sum_{i\geq 1}\sum_{e\in M^*_i}\frac{kn}{w_{\max}} w_G(e)$
$= \frac{w_G(M^*-M^*_0)}{1+\sigma}$.

Now, by Lemma 1, $w_G(M') \geq \frac{w_G(M^*-M^*_0)}{1+\sigma} \geq \frac{1}{1+\sigma}\left(1-\frac{1}{2k}\right) w_G(M^*)$. Clearly, $\frac{1}{1+\sigma}\left(1-\frac{1}{2k}\right) \geq 1-\frac{1}{\ell}$. This completes the proof. □

3.3 NC algorithm

In this section, we parallelize the greedy algorithm to obtain an NC approximation algorithm for the MWM problem that achieves a ratio of $\frac{1}{2+\epsilon}$ for any fixed $\epsilon > 0$.

The NC approximation algorithm will work on graph G' defined in Section 3.1. Recall that each edge e of G' inherits a rank $r(e)$ from G. Let $\ell = \lceil \log_{1+\sigma} kn \rceil$. The highest rank is ℓ. The algorithm works as follows:

NC Algorithm

1. For each $i = \ell, \ell-1, \cdots, 1$, perform the following steps:
   1.1. Find a maximal matching M_i in $G'_i = (V, F_i)$, where F_i is the set of edges of G' with rank i.
   1.2. Remove all edges e from G' such that $e \in M_i$ or e is incident to an endpoint of an edge in M_i.
2. Output $\cup_{1\leq i\leq \ell} M_i$.

Let $M = \cup_{1\leq i\leq \ell} M_i$. By Steps 1.1 and 1.2, M is clearly a maximal matching of G'.

Lemma 3. $w_G(M) \geq \frac{1}{2(1+\sigma)} w_G(M^* - M^*_0)$.

Proof. The idea is to distribute the weights of all edges of $M^* - M^*_0$ to the edges of M. Let e be an edge in $M^* - M^*_0$. Let $i = r(e)$. We distribute the weight $w_G(e)$ of e as follows:

- Case (1): $e \in M_i$. Then, we distribute $w_G(e)$ to e itself.
- Case (2): $e \notin M_i$ but one or both endpoints of e are incident to an edge of M_i. Then, we distribute $w_G(e)$ to an arbitrary edge $e' \in M_i$ such that e and e' share an endpoint. Note that $\frac{1}{1+\sigma} \leq \frac{w_G(e)}{w_G(e')} \leq 1+\sigma$.
- Case (3): None of cases (1) and (2) occurs. Then, by the algorithm, e must share an endpoint with at least one edge $e' \in M_j$ such that $j > i$. We distribute $w_G(e)$ to an arbitrary such edge e'. Note that $w_G(e) \leq (1+\sigma)^i \leq (1+\sigma)^{j-1} < w_G(e')$.

Consider an edge $e' \in M$. Since $M^* - M_0^*$ is a matching, at most two edges $e \in M^* - M_0^*$ can distribute their weights to e'. Moreover, by Cases (1) through (3), the total weight newly distributed to e' is at most $2(1+\sigma)w_G(e')$. Thus, $\sum_{e' \in M} 2(1+\sigma)w_G(e') \geq w_G(M^* - M_0^*)$. Consequently, $w_G(M) \geq \frac{1}{2(1+\sigma)} w_G(M^* - M_0^*)$. □

Thus we have the following theorem.

Theorem 2. *There is an* NC *approximation algorithm for the MWM problem that achieves a ratio of* $\frac{1}{2+\epsilon}$ *for any fixed* $\epsilon > 0$. *It runs in* $O(\log^4 n)$ *time using* $n + m$ *processors on the PRIORITY PRAM.*

Proof. Fix a positive real number ϵ. Suppose we want to compute a matching M of G with $w_G(M) \geq \frac{1}{2+\epsilon} \cdot w_G(M^*)$. Then, we fix $\sigma = \frac{\epsilon}{3}$ and $k = \lceil \frac{3}{\epsilon} + 1.5 \rceil$, construct G' as in Section 3.1, and run the above NC algorithm on input G' to obtain a matching M. By Lemmas 1 and 3, $w_G(M) \geq \frac{1}{2(1+\sigma)} \left(1 - \frac{1}{2k}\right) w_G(M^*) \geq \frac{1}{2+\epsilon} w_G(M^*)$.

We next analyze the complexity needed to compute M. G' can be constructed from G in $O(1)$ time using $n + m$ processors. M can be computed from G' in $O(\log_{1+\sigma}(kn) \cdot (\log n + T(n, m))$ time using $\max\{(n + m), P(n, m)\}$ processors on the PRIORITY PRAM, provided that a maximal matching of a given n-vertex m-edge graph can be computed in $T(n, m)$ using $P(n, m)$ processors on the PRIORITY PRAM. According to [IS86], $T(n, m) = \log^3 n$ and $P(n, m) = n + m$. So, M can be computed in $O(\log^4 n)$ time using $n + m$ processors on the PRIORITY PRAM. □

4 Approximation Algorithms for Graphs with Heavy Weights

We next show two algorithms that only use the total order of the weights. The first one is an NC algorithm, and the second one is an RNC algorithm. Both algorithms contain three phases:

Outline of Algorithms for heavy weights

1. For given G, construct a heavy spanning forest F (defined later) of G.
2. Construct a set of paths P in $G[F]$.
3. Produce a matching in $G[P]$.

The algorithms are the same except the phase 3. We now describe each phase, and analyze their complexity and approximation ratio.

4.1 The First Phase

The first phase contains two steps:

1.1. In parallel, each vertex marks the heaviest edge incident to the vertex;
1.2. F is the set of marked edges.

We first show that $G[F]$ is acyclic.

Proposition 1. *$G[F]$ is acyclic.*

Proof. Assume $G[F]$ is not acyclic and $e_1, e_2, \cdots, e_l$ are edges producing a cycle in $G[F]$. We let $v_1, v_2, \cdots, v_l$ be vertices on the cycle. If two consecutive vertices mark the same edge, there should be an edge on the cycle not marked by any vertices. Hence each vertex marks different edge. Thus we can assume that v_i marks e_i, and $w(e_1) < w(e_2)$. However, this implies that $w(e_1) < w(e_2) < \cdots < w(e_l) < w(e_1)$, that is a contradiction. □

Thus, $G[F]$ is a set of trees. Moreover, it is trivial that $\deg_{G[F]}(v) > 0$ for all v in V. Hence we call F *heavy spanning forest* of G. We now introduce some notions for the heavy spanning forest F. Let T be a tree in $G[F]$, and n_T be the number of vertices in T. Then, in the first step, each of n_T vertices in T marks one edge, and T has $n_T - 1$ edges. This implies that T has exactly one edge marked by its both endpoints. We call the edge and two vertices *a root edge* and *two roots* of T, respectively. That is, each tree has one root edge and two roots. We can show the following lemma by a simple induction.

Lemma 4. *Let T be a tree in F, and e_r be the root edge of T. Then for any leaf-root path $(e_l, e_1, e_2, \cdots, e_r)$ in T, $w(e_l) < w(e_1) < w(e_2) < \cdots < w(e_r)$.*

That is, the root edge is the heaviest edge in the tree. We now show the theorem for the relation between $w(M^*)$ and $w(F)$.

Theorem 3. *$w(F) \geq w(M^*)$.*

Proof. Let $e = \{u, v\}$ be an edge in M^*, but not in F. Then, since $e \notin F$, e is not marked by both u and v. Let e_u and e_v be edges marked by u and v, respectively. Since M^* is a matching, neither e_u nor e_v is not in M^*. That is, $\{e_u, e_v\} \subseteq F - M^*$. Now we divide the weight $w(e)$ in two weights $\frac{1}{2}w(e)$, and distribute them to e_u and e_v, respectively. Since e is not marked, $w(e) < w(e_u), w(e_v)$. Moreover, e_u and e_v are not distributed by the other edges in M^* at the points u and v since e is an element in the matching M^*. That is, no edge e' in $F - M^*$ will be distributed more than $w(e')$ by the edges in $M^* - F$. Thus each edge in M^* is either in F or it can be divided and distributed to two edges in $F - M^*$. This implies that $w(F) \geq w(M^*)$. □

We moreover analyze the proof of Theorem 3 in detail. Let $C = F \cap M^*$, $\hat{F} = F - C$, and $\hat{M} = M^* - C$. We let R be the set of the root edges of F. Then we have the following corollary.

Corollary 1. *$w(F) \geq 2w(M^*) - w(C) - w(R)$.*

Proof. In the proof of Theorem 3, each weight of an edge in $\hat{M}$ is *divided* and distributed to two edges in $\hat{F}$, because corresponding edges in $\hat{F}$ can be distributed at both endpoints. However, only root edges can be distributed at both

endpoints. Now we distribute each weight of an edge in $\hat{M}$ onto two edges in $\hat{F}$ without division. In the case, root edges may be distributed twice, hence we have $w(\hat{F}) \geq 2w(\hat{M}) - w(R)$. Thus $w(F) = w(\hat{F}) + w(C) \geq 2w(\hat{M}) - w(R) + w(C) = 2w(M^* - C) - w(R) + w(C) = 2w(M^*) - w(C) - w(R)$. □

4.2 The Second Phase

2.1. In parallel, each vertex v with $\deg_{G[F]}(v) > 2$ deletes all edges incident to v except two heaviest edges. Let P be the set of remaining edges. (Comment: It is easy to see that $G[P]$ is a set of paths, and $\deg_{G[P]}(v) > 0$ still holds if v was not a leaf in F.)

We show a proposition and a lemma for trees, and main theorem in this subsection.

Proposition 2. *Let T be a tree with n vertices. We let n_i be the number of vertices of degree i with $1 \leq i \leq \Delta_T$. Then (a) $\sum_{i=1}^{\Delta_T} n_i = n$; and (b) $\sum_{i=1}^{\Delta_T} in_i = 2(n-1)$.*

Proof. (a) is trivial. When each vertex counts up its degree, each edge is counted exactly twice. Since any tree with n vertices has $n-1$ edges [Har72, Chapter 4], (b) follows. □

Lemma 5. *Let $L(n, \Delta)$ be the maximum number of leaves in a tree of maximum degree Δ with n vertices. Then $L(n, \Delta) \leq \frac{(\Delta-2)n-2}{\Delta-1}$.*

Proof. Let T be an n vertex tree with $L(n, \Delta)$ leaves. Let V_I be the set of internal vertex of T. Then, the graph induced by V_I is also a tree. The induced tree contains $|V_I|$ vertices and $|V_I| - 1$ edges. Each leaf of T is incident to one internal vertex. Moreover, each vertex in V_I can be incident to at most Δ vertices. Thus we have $L(n, \Delta) \leq \Delta |V_I| - 2(|V_I| - 1)$. We also have $L(n, \Delta) + |V_I| = n$. Hence $L(n, \Delta) \leq \frac{(\Delta-2)n-2}{\Delta-1}$. □

Theorem 4. $w(P) > \frac{1}{\Delta_{G[F]}-1} w(F)$.

Proof. We first assume that $G[F]$ contains only one tree. We let n_i be the number of vertices of degree i in $G[F]$ with $1 \leq i \leq \Delta$. Each vertex does not delete the nearest edge to the root edge. Thus no edge will be deleted by two different vertices. Hence, by Proposition 2, the number of edges deleted in step 2.1 is equal to $\sum_{i=3}^{\Delta}(i-2)n_i = \sum_{i=3}^{\Delta} in_i - 2\sum_{i=3}^{\Delta} n_i = 2(n-1) - n_1 - 2n_2 - 2(n - n_1 - n_2) = n_1 - 2$. Using Lemma 5, we have $\frac{|P|}{|F|} = \frac{n-1-n_1+2}{n-1} \geq \frac{n+1-L(n,\Delta)}{n-1} \geq \frac{(\Delta-1)(n+1)-(\Delta-2)n+2}{(n-1)(\Delta-1)} = \frac{n+\Delta+1}{(n-1)(\Delta-1)} = \frac{1}{\Delta-1} + \frac{\Delta+2}{(n-1)(\Delta-1)} > \frac{1}{\Delta-1}$.

We then consider the weights of deleted edges. For each deleted edge e, there exists at least one edge e' in P with $w(e') > w(e)$. On the other hand, for each

remaining edge e' in P, it is corresponded by such deleted edges e at most $\Delta-1$ times. This together with $\frac{|P|}{|F|} > \frac{1}{\Delta-1}$ implies that $\frac{w(P)}{w(F)} > \frac{1}{\Delta-1}$.

When $G[F]$ contains two or more trees, the discussion above can be applied on each tree. More precisely, let F contain k trees $T_1, T_2, \cdots, T_k$, and P_i be the path set obtained from T_i with $1 \leq i \leq k$. Using the discussion above, we have $\frac{w(P_i)}{w(T_i)} > \frac{1}{\Delta-1}$, consequently, $w(P_i) > \frac{w(T_i)}{\Delta-1}$ with $1 \leq i \leq k$. Thus $w(P) = \sum_{i=1}^{k} w(P_i) > \frac{1}{\Delta-1}\sum_{i=1}^{k} w(T_i) = \frac{1}{\Delta-1}w(F)$, consequently, $\frac{w(P)}{w(F)} > \frac{1}{\Delta-1}$. □

4.3 The Third Phase

NC Algorithm

We define the distance of edges to describe the third phase of NC algorithm. Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path of length l. Then the *distance* of e_i from e_1 on α, denoted by $D(e_i, e_1)$, is defined by $D(e_1, e_1) = 0$, and $D(e_i, e_1) = D(e_{i-1}, e_1) + 1$ for $1 < i \leq l$. The third phase of NC algorithm contains the following steps:

3.1. In parallel, find the heaviest edge in each path in P.
3.2. M_N is the set of edges having even distance from the heaviest edge on the same path.

As a result, each path in P becomes alternating path containing the heaviest edge on the path admitted by M_N. We here show a proposition and a useful lemma for paths with special properties.

Proposition 3. *Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path with $w(e_1) > w(e_2) > \cdots > w(e_l)$. Then the maximum weighted matching of α, say M_α, is either $\{e_1, e_3, \cdots, e_l\}$ for odd l, or $\{e_1, e_3, \cdots, e_{l-1}\}$ for even l. Moreover, $w(M_\alpha) \geq \frac{1}{2}w(\alpha)$.*

Proof. When l is even, considering $w(e_1) > w(e_2)$, $w(e_3) > w(e_4)$, $\cdots$, $w(e_{l-1}) > w(e_l)$, we immediately have the proposition. When l is odd, the last edge e_l just increases the weight of M_α. □

Lemma 6. *Let $\alpha = (e_1, e_2, \cdots, e_l)$ be a path such that $w(e_1) < w(e_2) < \cdots < w(e_{i-1}) < w(e_i) > w(e_{i+1}) > \cdots > w(e_l)$ for some i with $1 < i < l$. Let A_1 be the alternating path containing e_i, A_2 be the other alternating path, and M_α be the maximum weighted matching of α.*

(1) Either $A_1 = M_\alpha$ or $A_2 = M_\alpha$. (Hence $w(M_\alpha) \geq \frac{1}{2}w(\alpha)$.)
(2) When $A_2 = M_\alpha$, $w(e_{i-1}) + w(e_{i+1}) \geq w(e_i)$.
(3) $w(A_1) \geq \frac{1}{3}w(\alpha)$.

Proof. (1) We first show that M_α satisfies either (a) $e_i \in M_\alpha$ or (b) $\{e_{i-1}, e_{i+1}\} \subseteq M_\alpha$. To derive a contradiction, assume that $e_i \notin M_\alpha$, $e_{i-1} \notin M_\alpha$, and $e_{i+1} \in M_\alpha$. Then $(M_\alpha - \{e_{i+1}\}) \cup \{e_i\}$ is a matching heavier than M_α since $w(e_i) > w(e_{i+1})$, that is a contradiction. The other symmetric case ($e_i \notin M_\alpha$, $e_{i-1} \in M_\alpha$, and

$e_{i+1} \notin M_\alpha$) can be shown by the same argument. In the case (a), we have $e_i \in M_\alpha$, $e_{i-1} \notin M_\alpha$, and $e_{i+1} \notin M_\alpha$. Then we can consider α as two paths $(e_1, e_2, \cdots, e_{i-2})$ and $(e_{i+2}, \cdots, e_l)$ separately. Using Proposition 3, we have $M_\alpha = \{e_i\} \cup \{e_{i-2}, e_{i-4}, \cdots\} \cup \{e_{i+2}, e_{i+4}, \cdots\}$, consequently, $M_\alpha = A_1$. In the case (b), we have $e_i \notin M_\alpha$, $e_{i-1} \in M_\alpha$, and $e_{i+1} \in M_\alpha$. Thus we have $M_\alpha = \{e_{i-1}, e_{i+1}\} \cup \{e_{i-3}, e_{i-5}, \cdots\} \cup \{e_{i+3}, e_{i+5}, \cdots\}$, consequently, $M_\alpha = A_2$.
(2) Let A_2' be $(A_2 - \{e_{i-1}, e_{i+1}\}) \cup \{e_i\}$. Then A_2' is a matching. Since A_2 is the maximum weighted matching, $w(A_2') \leq w(A_2)$. This implies that $w(e_{i-1}) + w(e_{i+1}) \geq w(e_i)$.
(3) By (1), either $A_1 = M_\alpha$ or $A_2 = M_\alpha$. When $A_1 = M_\alpha$, the claim follows from Proposition 3. Thus we assume that $A_2 = M_\alpha$.

We here consider two paths $\alpha_1 = (e_1, e_2, \cdots, e_{i-1}, e_i)$ and $\alpha_2 = (e_i, e_{i+1}, \cdots, e_l)$. Let A_1^1 (and A_1^2) be the alternating path of α_1 (and α_2, resp.) containing e_i. That is, A_1^1 is the former half of A_1, A_1^2 is the latter half of A_1, and $A_1^1 \cap A_1^2 = \{e_i\}$. Then, by Proposition 3, $w(A_1^1) \geq \frac{1}{2} w(\alpha_1)$ and $w(A_1^2) \geq \frac{1}{2} w(\alpha_2)$.

Thus, $w(A_1) = w(A_1^1 \cup A_1^2) = w(A_1^1) + w(A_1^2) - w(A_1^1 \cap A_1^2) \geq \frac{1}{2}(w(\alpha_1) + w(\alpha_2)) - w(e_i) = \frac{1}{2}(w(\alpha) + w(e_i)) - w(e_i) = \frac{1}{2}(w(\alpha) - w(e_i)) \geq \frac{1}{2}(w(\alpha) - w(A_1))$. This implies that $w(A_1) \geq \frac{1}{3} w(\alpha)$. □

We here remark that, in Lemma 6(1), we cannot determine which alternating path is heavier in general. (For example, a path (e_1, e_2, e_3) has different answers when $w(e_1) = 1, w(e_2) = 3, w(e_3) = 1$ and $w(e_1) = 2, w(e_2) = 3, w(e_3) = 2$). We also remark that Lemma 6(1) does not hold for general weighted path (for example, each alternating path of (e_1, e_2, e_3, e_4) is not the maximum weighted matching for $w(e_1) = 5, w(e_2) = 1, w(e_3) = 1, w(e_4) = 5$).

We now show the relation between $w(M_N)$ and $w(P)$.

Lemma 7. $w(M_N) \geq \frac{1}{3} w(P)$.

Proof. We first observe the following claim: in step 2.1, if vertex v delete an edge $\{u, v\}$, the edge was marked by u in step 1.1. This is easy because each vertex marked the heaviest edge in step 1.1, and remains the heaviest edge(s) in step 2.1. Using the claim and simple induction, we can show that each path in P is either

(1) a part of some leaf-root path in some tree in F; or
(2) two leaf-root paths connected by the root edge in a tree in F.

In the case (1), combining Lemma 4 and Proposition 3, M_N contains the maximum weighted matching of the path, and that has at least half weight of the path. Thus it is sufficient to show for the case (2). By Lemma 4, the path $\alpha = (e_1, e_2, \cdots, e_l)$ satisfies that $w(e_1) < w(e_2) < \cdots < w(e_{r-1}) < w(e_r) > w(e_{r+1}) > \cdots > w(e_l)$, where e_r is the root edge. Thus, according to Lemma 6(3), for the alternating path A_r containing e_r, $w(A_r) \geq \frac{1}{3} w(\alpha)$. Thus $w(M_N) \geq \frac{1}{3} w(P)$. □

Combining Theorem 3, Theorem 4, and Lemma 7, we can show that the NC algorithm is a $\frac{1}{3(\Delta-1)}$-approximation algorithm. But the better approximation ratio $\frac{2}{3\Delta+2}$ will be stated in Section 4.5.

RNC Algorithm

Phases 1 and 2 are performed "locally". That is, all computations can be performed within the neighbors. Using randomization, RNC algorithm finds a matching in $G[P]$ locally. The third phase of the RNC algorithm contains the following steps:

3.1'. In parallel, each vertex randomly choose one of two edges incident to the vertex in $G[P]$. (The vertices of degree one choose the unique edge incident to the vertex.)
3.2'. M_R is the set of edges chosen by both endpoints.

Since each vertex choose one edge, the resulting M_R is a matching. Moreover, since each edge in P is chosen with probability at least $\frac{1}{4}$, we immediately have the following lemma.

Lemma 8. *The expected value of $w(M_R)$ is at least $\frac{1}{4}w(P)$.*

4.4 Complexity of Algorithms

Each algorithm uses n processors; every vertex in G has a processor associated with it. As the input representation of G, we assume that each vertex has a list of the edges incident to it. Thus, each edge $\{i, j\}$ has two copies - one in the edge list for vertex i and the other in the edge list for vertex j.

Theorem 5. *The* NC *algorithm runs in $O(\log n)$ time using n processors on the PRIORITY PRAM. The algorithm only requires the total order of the weights.*

Proof. Each processor uses two memory cells to store the edges in P. The first and second phases can be efficiently implemented modifying as follows:

1.1'. In parallel, each vertex v finds the heaviest edge $e = \{v, u\}$ incident to v;
1.2'. In parallel, v stores the first cell of v with e.
2.1'. In parallel, each vertex v checks the contents of the first cell of u. If it is e, then the process is end. If it is not e, v tries to store the second cell of u with e. This trial will succeed if $w(e)$ is the heaviest among the other edges that are tried to store the same cell.

The step 1.2' can be done in a unit time. Moreover, it is not difficult to see that the steps 1.1' and 2.1' can be done in $O(\log \Delta)$ time using standard technique with comparison operation.

In the third phase, we can easy to see the following:
(1) if $e = \{u, v\}$ is a root edge, e is stored in the first cells of both u and v; and (2) otherwise, e is stored in the first cell of one endpoint, and in the second cell of the other.

Moreover, each non-root edge knows which endpoint is close to the root edge; the endpoint storing the second cell with the edge. Thus step 3.1 can be done in $O(1)$ time, and step 3.2 can be done in $O(\log n)$ time using standard list ranking technique (see e.g., [KR90]).

Throughout the computation, the algorithm only compares two weights of edges. Thus the algorithm only requires to know the total order of the weights. □

The third phase of the RNC algorithm can be performed in $O(1)$ time. This immediately implies the following theorem.

Theorem 6. *The* RNC *approximation algorithm runs in* $O(\log \Delta)$ *time using* n *processors on the PRIORITY PRAM. The algorithm only requires the total order of the weights.*

4.5 Approximation Ratios

We remind that M^* is a maximum weighted matching, F is the heavy spanning forest, R is the set of the root edges of F, and P is a set of paths obtained in step 2.1. Moreover we let $C = F \cap M^*$, $\hat{F} = F - C$, and $\hat{M} = M^* - C$.

To derive good approximation ratios, we define two maximum matchings: M_P denotes a maximum weighted matching of $G[P]$, and M_F denotes a maximum weighted matching of $G[F]$.

Lemma 9. $w(M_P) \geq \frac{1}{2(\Delta-1)} w(F)$.

Proof. As seen in the proof of Lemma 7, each path in P is either

(1) a part of some leaf-root path in some tree in F; or
(2) two leaf-root paths connected by a root edge in a tree in F.

For each path, by Lemma 6(1), M_P contains heavier alternating path that has at least half weight of the path. This together with Theorem 4 implies that $w(M_P) \geq \frac{1}{2} w(P) \geq \frac{1}{2(\Delta-1)} w(F)$. □

Lemma 10. $w(M_F) \geq \frac{1}{\Delta} w(M^*)$.

Proof. We first remind that M_F is the maximum weighted matching in F. Thus, since C is a matching in F, $w(M_F) \geq w(C)$. It is easy to see that R is a matching in F. This implies that $w(M_F) \geq w(R)$. It is also easy to see that M_P is a matching in F, and thus $w(M_F) \geq w(M_P)$. Hence, combining Corollary 1, we have $w(F) \geq 2w(M^*) - w(C) - w(R) \geq 2w(M^*) - 2w(M_F)$. On the other hand, by Lemma 9, $w(M_F) \geq w(M_P) \geq \frac{1}{2(\Delta-1)} w(F)$. Combining the equations, we have $(2(\Delta - 1) + 2)w(M_F) \geq 2w(M^*)$, consequently, $w(M_F) \geq \frac{1}{\Delta} w(M^*)$. □

Lemma 11. $w(C) \leq w(M_F) \leq 2w(M_P)$.

Proof. It is clear that $w(C) \leq w(M_F)$. Thus we show $w(M_F) \leq 2w(M_P)$. We are going to show that the weight of each edge in M_F can be distributed to an edge in M_P, and the weight of each edge in M_P is distributed by such edges at most twice. Let $e = \{u, v\}$ be any edge in M_F. Then three cases occur according to e.
(1) $e \in M_P$. We distribute $w(e)$ to itself.
(2) $e \in P - M_P$. We first assume that e is not a root edge. We assume that u is closer to the root edge than v on $G[F]$. In the case, e is incident to e' in P at the vertex u with $w(e') > w(e)$. Thus we distribute $w(e)$ to e'. We next assume that e is a root edge. That is, e is a root edge not in the maximum weighted matching of $G[P]$. Then, by Lemma 6, M_P contains two edges e' and e'' such that e' and e'' are the edges incident to e at vertex u and v, respectively, and $w(e')+w(e'') \geq w(e)$. Thus we divide $w(e)$ into $w(e')$ and $w(e)-w(e')(\leq w(e''))$, and distribute them to e' and e'', respectively.
(3) $e \notin M_P$. We assume that u is closer to the root edge than v on $G[F]$. In the case, e was deleted by u in step 2.1. The vertex u remains two edges e' and e'' in P with $w(e'), w(e'') > w(e)$. Moreover, either e' or e'' is in M_P. Thus we distribute $w(e)$ to the edge in M_P.

Since M_F is a matching, no two edges are distributed at the same endpoint. Thus each edge in M_P is distributed at most twice at both endpoints. This implies that $w(M_F) \leq 2w(M_P)$. □

Theorem 7. $w(M_P) \geq \frac{2}{2\Delta+1} w(M^*)$.

Proof. Combining Corollary 1 and Lemma 9, we get $w(M_P) \geq \frac{1}{2(\Delta-1)} w(F) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(R) - w(C))$. Using Lemma 11, we have $2w(M_P) \geq w(C)$. On the other hand, since $R \subseteq M_N$, $w(M_P) \geq w(M_N) \geq w(R)$. Thus, $w(M_P) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(R) - w(C)) \geq \frac{1}{2(\Delta-1)}(2w(M^*) - w(M_P) - 2w(M_P))$. Thus $w(M_P) \geq \frac{2}{2\Delta+1} w(M^*)$. □

Theorem 8. *The approximation ratio of the* NC *algorithm is* $\frac{2}{3\Delta+2}$.

Proof. We first show that $w(M_N) \geq \frac{1}{2} w(M_P)$. As seen in the proof of Lemma 7, each path in P is either

(1) a part of some leaf-root path in some tree in F; or
(2) two leaf-root paths connected by a root edge in a tree in F.

In the case (1), both M_N and M_P contain the same alternating path that contains the heaviest edge. We consider the paths in the case (2). Let α be the path in P, and A_1 be the alternating path of α containing the root edge, and A_2 be the other alternating path. According to Lemma 6, A_1 or A_2 is the maximum weighted matching of α. When A_1 is the maximum weighted matching, both M_N

and M_P contain it. Now we assume that A_2 is the maximum weighted matching of α. Then, by Lemma 6(3), $w(A_1) \geq \frac{1}{3}w(\alpha)$, consequently, $w(A_2) \leq \frac{2}{3}w(\alpha)$. Thus $w(A_1) \geq \frac{1}{2}w(A_2)$. Therefore, in any cases, $w(M_N) \geq \frac{1}{2}w(M_P)$.

Combining Corollary 1, Theorem 4, and Lemma 7, we have $w(M_N) \geq \frac{1}{3}w(P) \geq \frac{1}{3(\Delta-1)}w(F) \geq \frac{1}{3(\Delta-1)}(2w(M^*)-w(C)-w(R))$. It is clear that $w(M_N) \geq w(R)$ since $R \subseteq M_N$. Thus, using Lemma 11, we have $w(M_N) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - w(C) - w(R)) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - 2w(M_P) - w(M_N)) \geq \frac{1}{3(\Delta-1)}(2w(M^*) - 5w(M_N))$, consequently, $w(M_N) \geq \frac{2}{3\Delta+2}w(M^*)$. □

Theorem 9. *The approximation ratio of the* RNC *algorithm is* $\frac{1}{2\Delta+4}$.

Proof. Using Corollary 1, Theorem 4, and Lemma 8, we have $E(w(M_R)) \geq \frac{1}{4(\Delta-1)}w(F) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - w(C) - w(R))$.

We now compare $w(M_R)$ with $w(M_P)$. Each edge in M_P appears in M_R with probability at least $\frac{1}{4}$. This implies that the expected value of $w(M_R)$ is at least $\frac{1}{4}w(M_P)$. Thus, using Lemma 11, we have $E(w(M_R)) \geq \frac{1}{4(\Delta-1)}(2w(M^*) - w(C)-w(R)) \geq \frac{1}{4(\Delta-1)}(2w(M^*)-3w(M_P)) \geq \frac{1}{4(\Delta-1)}(2w(M^*)-12E(w(M_R)))$, consequently, $E(w(M_R)) \geq \frac{1}{2\Delta+4}w(M^*)$. □

References

[Avi83] D. Avis. A Survey of Heuristics for the Weighted Matching Problem. *Networks*, 13:475–493, 1983.

[Edm65] J. Edmonds. Paths, Trees and Flowers. *Canad. J. Math.*, 17:449–467, 1965.

[FGHP93] T. Fischer, A.V. Goldberg, D.J. Haglin, and S. Plotkin. Approximating matchings in parallel. *Information Processing Letters*, 46:115–118, 1993.

[Gab90] H.N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proc. 1st Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 434–443. ACM, 1990.

[Gal86] Z. Galil. Efficient Algorithms for Finding Maximum Matching in Graphs. *Computing Surveys*, 18(1):23–38, 1986.

[GHR95] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo. *Limits to Parallel Computation.* Oxford University Press, 1995.

[Har72] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[IS86] A. Israeli and Y. Shiloach. An Improved Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22:57–60, 1986.

[Joh90] D.S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, pages 69–161. Elsevier, 1990.

[KR90] R.M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, pages 870–941. Elsevier, 1990.

[KR98] M. Karpinski and W. Rytter. *Fast Parallel Algorithms for Graph Matching Problems.* Clarendon Press, 1998.

[KUW86] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a Perfect Matching is in Random NC. *Combinatorica*, 6(1):35–48, 1986.

[MS92] E.W. Mayr and A. Subramanian. The Complexity of Circuit Value and Network Stability. *Journal of Computer and System Science*, 44:302–323, 1992.

[Pre99] R. Preis. Linear Time $\frac{1}{2}$-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *STACS '99*, pages 259–269. Lecture Notes in Computer Science Vol. 1563, Springer-Verlag, 1999.

[Ven87] S.M. Venkatesan. Approximation Algorithms for Weighted Matching. *Theoretical Computer Science*, 54:129–137, 1987.

It Is on the Boundary: Complexity Considerations for Polynomial Ideals

Ernst W. Mayr

Lehrstuhl für Effiziente Algorithmen
Institut für Informatik
Technische Universität München, Germany
mayr@in.tum.de

Abstract. Systems of (in general non-linear but) polynomial equations over some ring or field R occur in numerous situations when dealing with problems in modelling, simulation, geometric representation and analysis, deduction, symbolic algebra, or dynamical systems, to name just a few. Algebraic varieties are determined by such systems (say over the reals or the complex number field), but they have also uses for propositional proof systems or for the modelling of certain parallel processes (as with reversible Petri nets).

Many, if not most of the questions concerning such systems of polynomial equations reduce to the investigation of properties of polynomial ideals in multi-variate polynomial rings (like $\mathbf{Q}[x_1, \ldots, x_n]$ or $\mathbf{Z}_2[x_1, \ldots, x_n]$), and earlier results have shown that such questions are generally very hard in the algorithmic sense, namely complete for the complexity class EXPSPACE.

As it turns out the algorithmic complexity of questions about polynomial ideals is really determined (as one might expect, of course) by the properties of the sets of exponent vectors occurring in the non-zero terms of the polynomials, and thus by the properties of seemingly well-structured subsets of $\mathbf{N}^n$, the nonnegative orthant of $\mathbf{Z}^n$. The algorithmic properties of such subsets have been studied extensively in numerous areas, as mentioned above, and their complexity has consistently been misjudged, based on the fact that "far out", i.e., for sufficiently large values of the coordinates, most of the relevant algorithmic problems become easy (NP or even P).

Here, we discuss how properties at the boundary of $\mathbf{N}^n$ (to $\mathbf{Z}^n$) affect the algorithmic complexity of basic questions about polynomial ideals. We also present some new algorithmic and complexity theoretic results based on ideal dimension and other structural properties, like for toric ideals.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, p. 99, 2000.
© Springer-Verlag Berlin Heidelberg 2000

An Efficient Parallel Algorithm for Scheduling Interval Ordered Tasks

Yoojin Chung[1], Kunsoo Park[2⋆], and Hyuk-Chul Kwon[3⋆]

[1] Research Institute of Computer and Information & Communication, Pusan National University, Pusan, 609-735, Korea
chungyj@pusan.ac.kr
[2] School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea
kpark@theory.snu.ac.kr
[3] Division of Computer Science and Engineering, Pusan National University, Pusan, 609-735, Korea
hckwon@pusan.ac.kr

Abstract. We present an efficient parallel algorithm for scheduling n unit length tasks on m identical processors when the precedence graphs are interval orders. Our algorithm requires $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter. By choosing $v = \sqrt{n}$, we obtain an $O(\sqrt{n} \log n)$-time algorithm with $O(n^2)$ operations. For $v = n/\log n$, we have an $O(\log^2 n)$-time algorithm with $O(n^3/\log^2 n)$ operations. The previous solution takes $O(\log^2 n)$ time with $O(n^3 \log^2 n)$ operations on the CREW PRAM. Our improvement is mainly due to a reduction of the m-processor scheduling problem for interval orders to that of finding a maximum matching in a convex bipartite graph.

1 Introduction

The m-processor scheduling problem for a precedence graph $\boldsymbol{G}$ is defined as follows. An input graph $\boldsymbol{G}$ has n vertices each of which represents a task to be executed on any one of m identical processors. Each task requires exactly one unit of execution time on any processor. At any timestep at most one task can be executed by a processor. If there is a directed edge from task t to task t', then task t must be completed before task t' is started. An m-processor schedule for $\boldsymbol{G}$ specifies the timestep and the processor on which each task is to be executed. The length of a schedule is the number of timesteps in it. A solution to the problem is an optimal (i.e., shortest length) schedule for $\boldsymbol{G}$.

The m-processor scheduling problem for arbitrary precedence graphs has been studied extensively. When $m = 2$, there are polynomial-time algorithms for the problem [6,3,9,7], and when m is part of the input, the problem is known to be NP-hard [20]. When m is part of the input, several researchers have considered restrictions on the precedence graphs. Polynomial-time algorithms for the m-processor scheduling problem are known for the cases that the precedence graphs

⋆ This work was supported by the Brain Korea 21 Project.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 100–111, 2000.
© Springer-Verlag Berlin Heidelberg 2000

are trees [12] and interval orders [15]. A survey of results on other special cases of the problem can be found in [13].

In parallel computation, the two processor case has been studied mostly. When $m = 2$, Helmbold and Mayr [11] gave the first NC algorithm and Vazirani and Vazirani [21] presented an RNC algorithm. Jung, Serna and Spirakis [16] developed an $O(\log^2 n)$-time algorithm using $O(n^3 \log^2 n)$ operations on the CREW PRAM. When $m = 2$ and the precedence graphs are interval orders, Moitra and Johnson [18] and Chung, Park and Cho [2] gave NC algorithms, and the one in [2] requires $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter.

When m is part of the input and the precedence graphs are interval orders, Sunder and He [19] developed the first NC algorithm for the scheduling problem, which takes $O(\log^2 n)$ time using $O(n^5 \log^2 n)$ operations or $O(\log^3 n)$ time using $O(n^4 \log^3 n)$ operations on the priority CRCW PRAM. Mayr [14] gave an $O(\log^2 n)$-time algorithm using $O(n^3 \log^2 n)$ operations on the CREW PRAM.

In this paper, we present an efficient parallel algorithm for the m-processor scheduling problem when the precedence graphs are interval orders. Our algorithm takes $O(\log^2 v + (n \log n)/v)$ time using $O(nv^2 + n^2)$ operations on the CREW PRAM, where $v \leq n$ is a parameter. By choosing $v = \sqrt{n}$, we obtain an $O(\sqrt{n} \log n)$-time algorithm with $O(n^2)$ operations. For $v = n/\log n$, we have an $O(\log^2 n)$-time algorithm with $O(n^3/\log^2 n)$ operations.

We briefly compare Mayr's algorithm and ours. A parallel algorithm that computes the length of an optimal m-processor schedule for an interval order will be called an *m-LOS algorithm*. Mayr's algorithm basically consists of two parts. The first part uses an m-LOS algorithm to compute the lengths of optimal schedules, which takes $O(\log^2 n)$ time using $O(n^3 \log^2 n)$ operations on the CREW PRAM. The second part computes an actual scheduling, which takes $O(\log^2 n)$ time using $O(n^3 \log^2 n)$ operations on the CREW PRAM. Our algorithm also consists of two parts and its first part is an m-LOS algorithm, but our algorithm is quite different from Mayr's as follows.

- We give an efficient m-LOS algorithm that takes $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM by generalizing the techniques used for two-processor scheduling in [2].
- After computing the lengths of optimal schedules, we reduce the m-processor scheduling problem for interval orders to that of finding a maximum matching in a convex bipartite graph using the lengths to compute an actual scheduling. Therefore, the part of computing an actual scheduling in our algorithm takes $O(\log^2 n)$ time using $O(n \log^2 n)$ operations on the EREW PRAM.

The remainder of this paper is organized as follows. The next section gives basic definitions and a sequential scheduling algorithm. Section 3 describes the reduction of m-processor scheduling to maximum matching in a convex bipartite graph. Section 4 describes our efficient m-LOS algorithm.

2 Basic Definitions and Sequential Algorithm

In this section we describe basic definitions and a sequential m-processor scheduling algorithm. An instance of the m-processor scheduling problem is given by a precedence graph $\boldsymbol{G} = (V, E)$. A *precedence graph* is an acyclic and transitively closed digraph. Each vertex of $\boldsymbol{G}$ represents a task whose execution requires unit time on one of m identical processors. If there is a directed edge from task t to task t', then task t must be completed before task t' is started. In such a case, we call t a *predecessor* of t' and t' a *successor* of t. We use $\langle t, t' \rangle$ to denote a directed edge from t to t'. A *schedule* is a mapping from tasks to timesteps such that at most m tasks are mapped to each timestep and for every edge $\langle t, t' \rangle$, t is mapped to an earlier timestep than t'. The length of a schedule is the number of timesteps used. An optimal schedule is one with the shortest length.

Let $I = \{I_1, \ldots, I_n\}$ be a set of intervals with each interval I_i represented by $I_i.l$ and $I_i.r$, where $I_i.l$ and $I_i.r$ denote the left and right endpoints of interval I_i, respectively. Without loss of generality, we assume that all the endpoints are distinct. We also assume that the intervals are labeled in the increasing order of right endpoints, i.e., $I_1.r < I_2.r < \cdots < I_n.r$ because sorting can be done in $O(\log n)$ time using $O(n \log n)$ operations on the EREW PRAM [4]. Given a set I of n intervals, let $\boldsymbol{G_I} = (V, E)$ be a graph such that

- $V = I = \{I_1, I_2, \ldots, I_n\}$ and
- $E = \{\langle I_i, I_j \rangle \mid 1 \leq i, j \leq n \text{ and } I_i.r < I_j.l\}$.

Such a graph $\boldsymbol{G_I}$ is called an *interval order*. Note that $\boldsymbol{G_I}$ is a precedence graph. Given a set I of n intervals, the *interval graph* G_I is an undirected graph such that each vertex corresponds to an interval in I and two vertices are adjacent whenever the corresponding intervals have at least one point in common. Therefore, an interval graph G_I is a complement of the interval order $\boldsymbol{G_I}$. We say that two vertices are *independent* if they are not adjacent in a graph. Note that overlapping intervals are adjacent in G_I and they are independent of each other in $\boldsymbol{G_I}$. In what follows, we use the words *tasks* and *intervals* interchangeably.

A schedule of length r on m processors for an interval order $\boldsymbol{G_I}$ can be represented by an $m \times r$ matrix M, where the columns are indexed by $1, \ldots, r$ and the rows are indexed by $1, \ldots, m$. Let $P_1, \ldots, P_m$ denote the m identical processors. If task x is scheduled on processor P_i at timestep τ, then x is assigned to a slot $M[i, \tau]$. No two tasks are assigned to the same slot in M. A slot of M to which no task is assigned is said to have an *empty task*. We assume that the right endpoint of an empty task is larger than all right endpoints in I. A column of M is called *full* if it does not have an empty task. Let $opt(I)$ be the length of an optimal schedule for an interval order $\boldsymbol{G_I}$.

Algorithm m-seq(I, m)

Input: intervals in I
Output: $m \times opt(I)$ matrix M_s

begin
$\tau \leftarrow 1$;
$S_\tau \leftarrow$the list of intervals in I sorted in the increasing order of right endpoints;
while $S_\tau \neq \phi$ **do**
    $S' \leftarrow \{\}$;
    Extract the first interval from S_τ and insert it to S';
    repeat
        Scan S_τ from left to right. When interval w is scanned,
            if w is overlapping every interval in S'
            then extract w from S_τ and insert it to S' **fi**;
    until (S' contains m intervals **or** all intervals of S_τ are considered)
    Schedule the intervals of S' in column τ of M_s
        in the order of the elements in list S';
    $S_{\tau+1} \leftarrow S_\tau$;
    $\tau \leftarrow \tau + 1$;
od
Output the schedule M_s constructed;
end

Fig. 1. Sequential scheduling algorithm

The sequential algorithm [15] in Figure 1 solves the m-processor scheduling problem for an interval order $\boldsymbol{G_I}$, which runs in $O(n \log n)$ time. Let $I(1, j)$ denote $\{I_1, \ldots, I_j\}$, $1 \leq j \leq n$. Note that m-seq computes an optimal schedule for $\boldsymbol{G_{I(1,j)}}$. We can easily get the following facts from algorithm m-seq.

Fact 1 *All the intervals in the same column of M_s overlap each others.*

Fact 2 *In each column τ of M_s in* m-seq, $M_s[1, \tau].r \leq M_s[2, \tau].r \leq \ldots \leq M_s[m, \tau].r$.

Fact 3 *In the first row of M_s, $M_s[1, 1].r < M_s[1, 2].r < \ldots < M_s[1, m].r$.*

Proof. It follows from the fact that for every τ, $M_s[1, \tau]$ is the first ending interval in S_τ and $M_s[1, \tau']$ with $\tau' > \tau$ is in S_τ.

3 Constructing an Optimal Schedule

In this section we describe our parallel m-processor scheduling algorithm for interval orders. We first describe characteristics of maximal cliques in interval graphs. A set of intervals form a *clique* if each pair of intervals in the set has a nonempty intersection. If we scan any given interval x from its left endpoint to its right, we can meet all those maximal cliques to which x belongs. This yields the Gilmore-Hoffman theorem [10].

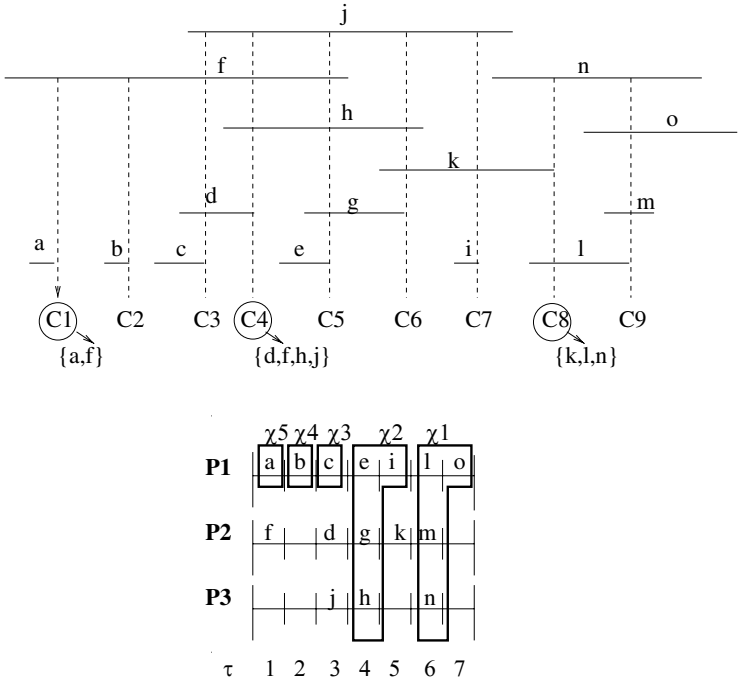


Fig. 2. An interval set I and $\boldsymbol{G_I}$'s optimal schedule when $m = 3$.

Theorem 1. [10] *The maximal cliques of an interval graph can be linearly ordered so that for any given interval x, the set of cliques in which x occurs appear consecutively in the linear order.*

Let k be the number of maximal cliques in G_I. Let $C_1, \ldots, C_k$ be the maximal cliques of G_I in the ordering of Theorem 1. Given an interval set, we can find the maximal cliques of the interval graph G_I using Lemma 1. In Figure 2, dotted vertical lines mark the right endpoints of Lemma 1, i.e., there are nine maximal cliques in G_I and they are $C_1 = \{a, f\}$, $C_2 = \{b, f\}$, $C_3 = \{c, d, f, j\}$, etc.

Lemma 1. [2] *In an interval set I, a right endpoint represents a maximal clique of G_I if and only if its previous endpoint in the sorted list of left and right endpoints is a left endpoint.*

For each interval $x \in I$, let s_x and l_x be the smallest and the largest j, respectively, such that x belongs to C_j. In Figure 2, $s_h = 4$ and $l_h = 6$ because interval h is in C_4, C_5 and C_6. Let $sltask(i, j)$, $1 \leq i, j \leq k$, be the set of intervals x such that $i \leq s_x$ and $l_x \leq j$. In Figure 2, $sltask(1, 5) = \{a, b, c, d, e, f\}$. Note that algorithm m-seq(I, m) in Figure 1 computes an optimal schedule for $\boldsymbol{G_{sltask(1,j)}}$, $1 \leq j \leq k$, because m-seq computes an optimal schedule for $\boldsymbol{G_{I(1,t)}}$, $1 \leq t \leq n$, and maximal cliques $C_1, \ldots, C_k$ of G_I are labeled by scanning endpoints of I

from left to right using Lemma 1. Let $len(i, j)$ be the minimum number of time-steps required to schedule all tasks in $sltask(i, j)$, i.e., $opt(sltask(i, j))$.

Lemma 2. [2] *For two intervals $x, y \in I$, $l_x < s_y$ if and only if $x.r < y.l$.*

We now describe our parallel m-processor scheduling algorithm for interval orders. Our algorithm consists of two parts. The first part is an m-LOS algorithm m-length, which will be described in Section 4. Algorithm m-length computes $len(1, j)$ for all $1 \leq j \leq k$. The second part computes an optimal schedule by reducing the m-processor scheduling problem for an interval order to that of finding a maximum matching in a convex bipartite graph.

We first describe the definition of a convex bipartite graph. A convex bipartite graph G is a triple (A, B, E) such that $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ are disjoint sets of vertices and the edge set E satisfies the following properties:

(1) Every edge of E is of the form (a_i, b_j).
(2) If $(a_i, b_j) \in E$ and $(a_i, b_{j+t}) \in E$, then $(a_i, b_{j+r}) \in E$ for every $1 \leq r < t$.

Property (1) is a bipartite property while property (2) is a convexity property. It is clear that every convex bipartite graph $G = (A, B, E)$, where $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$, is uniquely represented by a set of triples: $T = \{(a_i, g_i, h_i) \mid 1 \leq i \leq n\}$, where $g_i = \min\{j \mid (a_i, b_j) \in E\}$ and $h_i = \max\{j \mid (a_i, b_j) \in E\}$. Dekel and Sahni [5] developed an $O(\log^2 n)$-time convex bipartite maximum matching algorithm using $O(n \log^2 n)$ operations on the EREW PRAM.

Our m-processor scheduling algorithm is as follows.

Algorithm m-schedule

- Step 1: Compute s_x and l_x for every $x \in I$.
- Step 2: Let $L_0 = 0$. Let $L_j = len(1, j)$ for $1 \leq j \leq k$ and compute L_j.
- Step 3: Construct a convex bipartite graph $G_b = (A_b, B_b, E_b)$, where $A_b = I$, $B_b = \{1, 2, \ldots, mL_k\}$ and E_b is computed from L_j, $j \leq k$, as follows. If an interval $x \in I$ is in a maximal clique C_t in G_I, then x is adjacent to all j in B_b such that $mL_{t-1} + 1 \leq j \leq mL_t$. Since an interval x is in every C_t such that $s_x \leq t \leq l_x$ by Theorem 1, G_b is represented by $T = \{(x, mL_{s_x-1}+1, mL_{l_x}) \mid x \in I\}$.
- Step 4: Find a maximum matching in G_b. Then an optimal schedule for $\boldsymbol{G_I}$ is represented by an $m \times L_k$ matrix M_b, whose j-th column consists of the tasks in A_b matched with $m(j-1)+1, \ldots, mj$ in B_b in the maximum matching of G_b.

We now prove the correctness of algorithm m-schedule.

Lemma 3. *All the intervals in the same column of M_b are independent of each other in $\boldsymbol{G_I}$.*

Proof. By definition of G_b, all intervals that are adjacent to one of $mL_{j-1}+1,\ldots,mL_j$ in G_b, $1 \le j \le L_k$, are also adjacent to all of $mL_{j-1}+1,\ldots,mL_j$ and they are all in the same maximal clique in G_I. Therefore, the intervals matched with $mL_{j-1}+1,\ldots,mL_j$ in the maximum matching of G_b are independent of each other in $\boldsymbol{G_I}$. Since all the intervals in columns $L_{j-1}+1,\ldots,L_j$, $1 \le j \le k$, in M_b are independent of each other in $\boldsymbol{G_I}$, we have the lemma.

Lemma 4. *The convex bipartite graph $G_b = (A_b, B_b, E_b)$ has a maximum matching of size n, i.e., all intervals in A_b are matched in a maximum matching of G_b.*

Proof. Construct an edge set $E' \subseteq A_b \times B_b$ from M_s constructed by algorithm m-seq in Figure 1 as follows. $E' = \{(x,j) \mid x \in A_b$ is the j-th element of M_s in the column-major order$\}$. Then every edge (x,j) in E' satisfies $m(\tau-1)+1 \le j \le m\tau$, where τ is the column number in M_s at which x is. We first show that $E' \subseteq E_b$. Note that $\tau \le L_{l_x}$ because m-seq produces an optimal schedule for $\boldsymbol{G_{sltask(1,l_x)}}$. And we have $\tau > L_{s_x-1}$ by the following.

- If x is in the first row in M_s, then $\tau > L_{s_x-1}$ because $x \notin sltask(1, s_x-1)$ and the task in the first row uses a new time unit after time L_{s_x-1}.
- If x is in row r such that $r \ge 2$, i.e., $x = M_s[r,\tau]$, then $M_s[1,\tau] \notin sltask(1, s_x-1)$ because $M_s[1,\tau]$ and x overlap by Fact 1, and thus $\tau > L_{s_x-1}$.

Hence $L_{s_x-1}+1 \le \tau \le L_{l_x}$. Since x is adjacent to all t such that $mL_{s_x-1}+1 \le t \le mL_{l_x}$ in E_b, every edge (x,j) in E' is also in E_b. Since j's are distinct, E' is a maximum matching of size n in G_b.

Lemma 5. *The $m \times L_k$ matrix M_b is an optimal schedule for $\boldsymbol{G_I}$.*

Proof. Consider tasks x and y of $\boldsymbol{G_I}$ such that y is a successor of x. Let τ and τ' be the columns of M_b at which x and y are, respectively. Note that M_b has L_k columns, which is $opt(I)$, and all tasks are in M_b by Lemma 4. Since all the tasks in the same column of M_b are independent of each other in $\boldsymbol{G_I}$ by Lemma 3, we can prove that M_b is an optimal schedule for $\boldsymbol{G_I}$ by showing that $\tau' > \tau$.

Let t and t' be integers matched with x and y, respectively, in the maximum matching of G_b. Then $t \le mL_{l_x}$ and $mL_{s_y-1}+1 \le t'$ by definition of G_b. Since y is a successor of x, we have $l_x < s_y$ by Lemma 2, which implies that t' is greater than t. Since y must be in a different column of M_b with that of x by Lemma 3, we have $\tau' > \tau$.

Theorem 2. *An optimal schedule for $\boldsymbol{G_I}$ on m processors can be solved in $O(\log^2 v + (n \log n)/v)$ time with $O(nv^2+n^2)$ operations on the CREW PRAM, where $v \le n$ is a parameter.*

Proof. The correctness of algorithm m-schedule follows from Lemma 5. We will show that m-schedule takes $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations on the CREW PRAM. Step 1 takes $O(\log n)$ time using $O(n \log n)$ operations as follows. In a sorted endpoints sequence, put 1 at the right endpoints of Lemma 1 and 0 in other endpoints and compute s_x and l_x using a prefix sum, i.e., the prefix sum at $x.l$ is $s_x - 1$ and the prefix sum at $x.r$ is l_x. Since we can compute all L_j $(= len(1, j))$ for $1 \leq j \leq k$ by running algorithm m-length in Section 4 only once, Step 2 takes $O(\log^2 v + (n \log n)/v)$ time with $O(nv^2 + n^2)$ operations. Step 3 takes constant time using $O(n)$ operations. Step 4 takes $O(\log^2 n)$ time with $O(n \log^2 n)$ operations using Dekel and Sahni's algorithm [5].

4 Computing the Length of an Optimal Schedule

We now describe our m-LOS algorithm. We obtain our m-LOS algorithm in Figure 3 by generalizing the 2-LOS algorithm in [2].

Algorithm m-length
for all i, j with $1 \leq i \leq j \leq k$ **do in parallel**
  compute $|sltask(i, j)|$
  $len_0(i, j) = \lceil |sltask(i, j)|/m \rceil$
od
for $r = 1$ to $\lceil \log n \rceil$ **do**
  for all i, j with $1 \leq i \leq j \leq k$ **do in parallel**
    $len_r(i, j) = \max_{i \leq x \leq j} \{len_{r-1}(i, x) + len_{r-1}(x + 1, j)\}$
  od
od
print $len_{\lceil \log n \rceil}(1, k)$
end

Fig. 3. An efficient m-LOS algorithm

We now prove the correctness of algorithm m-length. We first define sets $\chi_1, \ldots, \chi_z$ of tasks for an interval order such that:

- all tasks in any χ_{i+1} are predecessors of all tasks in χ_i and
- the length of an optimal schedule equals $\sum_i \lceil |\chi_i|/m \rceil$.

Our sets χ_i's for m-processor scheduling are the generalization of those for two-processor scheduling [3] tailored to the special case of interval orders. We do not explicitly compute these sets in algorithm m-length in Figure 3; we only make use of them for the proof of its correctness.

We define the sets $\chi_1, \ldots, \chi_z$ of tasks from the schedule M_s computed by algorithm m-seq in Figure 1 as follows. We recursively define tasks v_i and w_i for $i \geq 1$. Let v_1 be the last task executed by processor P_1 (i.e., v_1 is $M_s[1, opt(I)]$) and w_1 is (a possibly empty task) $M_s[m, opt(I)]$. Given v_i, we define w_{i+1} and v_{i+1} as follows. Suppose that v_i is $M_s[1, \tau]$. Let τ' be the largest column number

less than τ in M_s such that $M_s[m, \tau'].r > v_i.r$ or $M_s[m, \tau']$ is an empty task. Then w_{i+1} is $M_s[m, \tau']$ and v_{i+1} is $M_s[1, \tau']$. In Figure 2, $v_1 = o$, and thus w_2 is an empty task and $v_2 = i$. Also $w_3 = j$ and $v_3 = c$. Note that each column τ'' such that $\tau' < \tau'' < \tau$ is full. Let z be the largest index for which w_z and v_z are defined. We assume that v_{z+1} is a special interval β whose right endpoint is smaller than all endpoints in I and $l_{v_{z+1}} = 0$. Let τ_i, $1 \leq i \leq z$, denote the timestep at which v_i is executed. Define χ_i to be $\{x | x$ is in column τ'' such that $\tau_{i+1} < \tau'' < \tau_i\} \cup \{v_i\}$. In Figure 2, sets χ_i's for $\boldsymbol{G_I}$ are marked by thick lines in the schedule. The characteristics of χ_i's are as follows.

Lemma 6. *In $\boldsymbol{G_I}$, every task $x \in \chi_i$ satisfies $x.r \leq v_i.r$.*

Proof. Since τ_{i+1} is the largest column number less than τ_i such that $M_s[m, \tau_{i+1}].r > v_i.r$, we have $M_s[m, \tau''].r < v_i.r$ for $\tau_{i+1} < \tau'' < \tau_i$. Note that we assume that an empty task has the largest right endpoint in I. Since the task in the last row in each column has the largest right endpoint in the column by Fact 2, every task x in column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ satisfies $x.r < v_i.r$. Therefore, every $x \in \chi_i$ satisfies $x.r \leq v_i.r$.

Lemma 7. *In $\boldsymbol{G_I}$, all tasks in χ_{i+1} are predecessors of all tasks in χ_i.*

Proof. Let y be a task in χ_i. Since every $x \in \chi_{i+1}$ satisfies $x.r \leq v_{i+1}.r$ by Lemma 6, we can prove the lemma by showing that $v_{i+1}.r < y.l$. Since $y.r \leq v_i.r$ and $v_i.r < w_{i+1}.r = M_s[m, \tau_{i+1}].r$, we have $y.r < M_s[m, \tau_{i+1}].r$. Since y is at one of columns $\tau_{i+1} + 1, \ldots, \tau_i$, we have $M_s[1, \tau_{i+1}].r < y.r$ by Facts 2 and 3. Hence $M_s[1, \tau_{i+1}].r < y.r < M_s[m, \tau_{i+1}].r$. If y overlaps $M_s[1, \tau_{i+1}] = v_{i+1}$, then y should be assigned to column τ_{i+1} in m-seq in Figure 1, which is a contradiction. Therefore, $v_{i+1}.r < y.l$.

Theorem 3. *The length of an optimal schedule for $\boldsymbol{G_I}$ is $\sum_{1 \leq i \leq z} \lceil |\chi_i|/m \rceil$.*

Proof. Since each column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ is full and $v_i = M_s[1, \tau_i]$ is in χ_i, we get $\lceil |\chi_i|/m \rceil = \tau - \tau'$. Therefore, $\sum_{1 \leq i \leq z} \lceil |\chi_i|/m \rceil$ is the number of columns in M_s, which is $opt(I)$.

When $m = 2$, Chung et al. [2] showed that χ_i equals $sltask(l_{v_{i+1}} + 1, l_{v_i})$ for $1 \leq i \leq z$ and that $len_{\lceil \log n \rceil}(i, j)$ equals $len(i, j)$ for $1 \leq i \leq j \leq k$. Similarly, we can prove the correctness of algorithm m-length as follows.

Lemma 8. *In $\boldsymbol{G_I}$, $\chi_i \subseteq sltask(l_{v_{i+1}} + 1, l_{v_i})$ for $1 \leq i \leq z$.*

Proof. Let x be a task in χ_i. Since $v_{i+1} \in \chi_{i+1}$ is a predecessor of x by Lemma 7, we have $v_{i+1}.r < x.l$, which implies $l_{v_{i+1}} < s_x$ by Lemma 2. Since $x.r \leq v_i.r$ by Lemma 6, we have $l_x \leq l_{v_i}$. Therefore, x is in $sltask(l_{v_{i+1}} + 1, l_{v_i})$.

Corollary 1. *In $\boldsymbol{G_I}$, $\bigcup_{i \leq t \leq j} \chi_t \subseteq sltask(l_{v_{j+1}} + 1, l_{v_i})$ for $1 \leq i \leq j \leq z$.*

Corollary 2. *In $\boldsymbol{G_I}$, all tasks in $sltask(l_{v_{j+1}}+1, l_{v_i})$, $i \le j$, are successors of all tasks in $\bigcup_{j+1\le t\le z}\chi_t$ and predecessors of all tasks in $\bigcup_{1\le t\le i-1}\chi_t$.*

Lemma 9. *Every task in $sltask(l_{v_{i+1}}+1, l_{v_i})$ is in one of columns $\tau_{i+1}+1, \ldots, \tau_i$.*

Proof. Let y be a task in $sltask(l_{v_{i+1}}+1, l_{v_i})$. Note that y satisfies $M_s[1, \tau_{i+1}].r < y.l$ by Lemma 2 and $y.r < M_s[1, \tau_i+1].l$ by Lemma 7. Therefore, y must be in one of columns $\tau_{i+1}+1, \ldots, \tau_i$ by the way algorithm m-seq in Figure 1 works.

Lemma 10. *In $\boldsymbol{G_I}$, $\sum_{i\le t\le j}\lceil|\chi_t|/m\rceil = len(l_{v_{j+1}}+1, l_{v_i})$ for $1 \le i \le j \le z$.*

Proof. The proof for the case $m = 2$ is in Lemma 8 in [2] and the proof of the lemma is similar.

Lemma 11. *In algorithm* m-length, *$len_r(i,j) \le len(i,j)$ for $0 \le r \le \lceil \log n\rceil$.*

Proof. It is similar to the proof of Lemma 9 in [2].

Lemma 12. *In algorithm* m-length, *$len_{\lceil \log n\rceil}(i,j) \ge len(i,j)$ for $1 \le i \le j \le k$.*

Proof. We show that $len_{\lceil \log n\rceil}(1,k) \ge len(1,k)$. We prove by induction on r that for $i \le 2^r$,

$$len_r(l_{v_{x+i}}+1, l_{v_x}) \ge \sum_{x\le t<x+i}\lceil|\chi_t|/m\rceil \tag{1}$$

When $r = 0$, (1) holds as follows. Since each column τ'' such that $\tau_{i+1} < \tau'' < \tau_i$ is full, $\lceil|sltask(l_{v_{x+1}}+1, l_{v_x})|/m\rceil \ge \lceil|\chi_x|/m\rceil \ge \tau_i - \tau_{i+1}$ by Lemma 8. Since $\lceil|sltask(l_{v_{x+1}}+1, l_{v_x})|/m\rceil \le \tau_i - \tau_{i+1}$ by Lemma 9, we have $\lceil|sltask(l_{v_{x+1}}+1, l_{v_x})|/m\rceil = \tau_i - \tau_{i+1}$. Therefore, $len_0(l_{v_{x+1}}+1, l_{v_x}) = \lceil|sltask(l_{v_{x+1}}+1, l_{v_x})|/m\rceil = \lceil|\chi_x|/m\rceil$. Assume that (1) holds after r iterations of the main loop. In the $(r+1)$st iteration for $2^r < i \le 2^{r+1}$,

$$\begin{aligned} len_{r+1}(l_{v_{x+i}}+1, l_{v_x}) &\ge len_r(l_{v_{x+i}}+1, l_{v_{x+2^r}}) + len_r(l_{v_{x+2^r}}+1, l_{v_x}) \\ &\ge \sum_{x+2^r\le t<x+i}\lceil|\chi_t|/m\rceil + \sum_{x\le t<x+2^r}\lceil|\chi_t|/m\rceil \\ &\ge \sum_{x\le t<x+i}\lceil|\chi_t|/m\rceil \end{aligned}$$

Since each χ_i contains at least one task, there are at most n χ_i's. Thus,

$$\begin{aligned} len_{\lceil \log n\rceil}(l_{v_{z+1}}+1, l_{v_1}) &\ge \sum_{1\le t\le z}\lceil|\chi_t|/m\rceil \\ &\ge len(l_{v_{z+1}}+1, l_{v_1}) \quad \text{by Lemma 10.} \end{aligned}$$

Since $l_{v_{z+1}}+1 = 1$ and $l_{v_1} = k$, we get $len_{\lceil \log n\rceil}(1,k) \ge len(1,k)$. Similarly, we can prove that $len_{\lceil \log n\rceil}(i,j) \ge len(i,j)$ for $1 \le i \le j \le k$ by using sets of χ_i's for $\boldsymbol{G_{I'}}$, where I' is $sltask(i,j)$ in I.

Theorem 4. *There is an m-LOS algorithm that requires* $O(\log^2 v + (n \log n)/v)$ *time and* $O(nv^2 + n^2)$ *operations on the CREW PRAM, where* v *is a parameter such that* $v \leq n$*. Furthermore, it also computes the length of an optimal schedule for* $\boldsymbol{G_{sltask(1,j)}}$*,* $1 \leq j \leq k$*.*

Proof. The correctness of algorithm m-length follows from Lemmas 11 and 12. Algorithm m-length has a straightforward implementation using $O(\log^2 n)$ time and $O(n^3)$ processors on the CREW PRAM. It can be improved to $O(\log^2 v + (n \log n)/v)$ time and $O(nv^2 + n^2)$ operations using Galil and Park's reduction technique [8], which is similar to the proof of Theorem 3 in [2].

References

1. M. Bartusch, R. H. Mohring, and F. J. Radermacher, "M-machine unit time scheduling: A report of ongoing research," *Lecture Notes in Economics and Mathematical Systems* **304**, Springer-Verlag (1988) 165–212.
2. Y. Chung, K. Park, and Y. Cho, "Parallel maximum matching algorithms in interval graphs," *Int'l. J. Foundations of Comput. Science* **10**, **1** (1999) 47–60.
3. E. Coffman and R. Graham, "Optimal scheduling for two processor systems," *Acta Informatica* **1** (1972) 200–213.
4. R. Cole, "Parallel merge sort," *SIAM J. Comput.* **17**, **4** (1988) 770–785.
5. E. Dekel and S. Sahni, "A parallel matching algorithm for convex bipartite graphs and applications to scheduling," *J. Parallel Distrib. Comput.* **1** (1984) 185–205.
6. M. Fujii, T.Kasami, and K. Ninomiya, "Optimal sequencing of two equivalent processors," *SIAM J. Appl. Math.* **17** (1969) 784–789.
7. H.N. Gabow, "An almost-linear algorithm for two-processor scheduling," *J. ACM* **29**, **3** (1982) 766–780.
8. Z. Galil and K. Park, "Parallel algorithms for dynamic programming recurrences with more than $O(1)$ dependency," *J. Parallel Distrib. Comput.* **21**, (1994) 213–222.
9. M. R. Garey and D. S. Johnson, "Scheduling tasks with nonuniform deadlines on two processors," *J. ACM* **23** (1976) 461–467.
10. M.C. Golumbic, *Graph theory and perfect graphs*, Academic Press, New York, 1980.
11. D. Helmbold and E. Mayr, "Two processor scheduling is in NC," *SIAM J. Comput.* **16**, **4** (1987) 747–759.
12. T. C. Hu, "Parallel sequencing and assembly line problems," *Oper. Res.* **9** (1961) 841–848.
13. E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Technical report*, Centrum voor Wiskunde en Informatica, 1989.
14. E. Mayr, "Scheduling interval orders in parallel," *Parallel Algorithms and Applications* **8** (1996) 21–34.
15. C. H. Papadimitriou and M Yannakakis, "Scheduling interval-ordered tasks," *SIAM J. Comput.* **8** (1979), pp. 405–409.
16. H. Jung, M. Serna and P. Spirakis, "A parallel algorithm for two processors precedence constraint scheduling," *Proc. Int'l Colloquium on Automata, Languages and Programming* (1991) pp. 417–428.
17. D. Kozen, U.V. Vazirani and V.V. Vazirani, "NC algorithms for cocomparability graphs, interval graphs, and unique perfect matchings," *Proc. Foundations of Software Technology and Theoretical Computer Science* (1985) pp. 496–503.

18. A. Moitra and R. Johnson, "A parallel algorithm for maximum matching on interval graphs," *Proc. Int'l Conference on Parallel Processing* **3** (1989) pp. 114–120.
19. S. Sunder and X.He, "Scheduling interval ordered tasks in parallel," *J. Algorithm* **26** (1998), pp.34–47.
20. J. D. Ulman, Complexity of sequencing problems, *in* "Computer and job scheduling theory" (E. G. Coffman, Ed.), Wiley, 1976.
21. U.V. Vazirani and V.V. Vazirani, "The two processor scheduling is in random NC," *SIAM J. Comput.* (1989) pp. 1140–1148.

Task Distributions on Multiprocessor Systems

Evgeny V. Shchepin[1⋆] and Nodari N. Vakhania[2⋆⋆]

[1] Steklov Math. Inst., 117966, Gubkina 8, Moscow
scepin@matcuer.unam.mx
[2] Faculty of Sciences
State University of Morelos
Av. Universidad 1001, Cuernavaca 62210, Mor. Mexico
nodari@servm.fc.uaem.mx

Abstract. We consider the problem of scheduling of n independent jobs on m unrelated machines to minimize the $\max(t_1, t_2, ..., t_m)$, t_i being the completion time of machine i. In [1] was suggested a polynomial 2-approximation algorithm for this problem. It was also proved that there can exist no polynomial 1.5-approximation algorithm unless $P = NP$. Here we improve this earlier performance bound 2 to $2 - \frac{1}{m}$. In [1] is also proved a general *rounding theorem*, which allows to construct in polynomial time 1-job approximations to the optimum, i.e. schedules with an absolute bound equal to the largest job processing time. We also improve this result and obtain $(1 - \frac{1}{m})$-job approximation to optimal.

Keywords: approximation algorithm, distribution, independent jobs, unrelated processors, makespan

1 Introduction

In this paper we consider one of the classical scheduling problems. We are given n tasks and m unrelated parallel processors. The processing time of a task on a processor is an arbitrary real number, quite independent from the processing time of any other task on that processor and from the processing time of this task on any other processor (this is in contrast with the situation with identical or uniform processors, when task processing times are more restricted). No task preemption is allowed, each machine can process at most one task at a time and we wish to minimize $\max(t_1, t_2, ..., t_m)$, t_i being the completion time of machine i. Even is $m = 2$ and the processors are identical, the problem is NP-hard [2]. Hence, no polynomial algorithm can build an optimal schedule for $m \geq 2$ processors, unless $P \neq NP$, and we try to approximate the optimum in polynomial time. For a given schedule, the optimality (or performance) ratio is defined as the ratio of the makespan of this schedule to the optimal makespan.

⋆ Partially supported by CONACYT grant 980066 and Russian Foundation of Basic Researches grant 99-01-00009
⋆⋆ Partially supported by CONACYT grant #473100-5-28937A

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 112–125, 2000.
© Springer-Verlag Berlin Heidelberg 2000

A schedule with the optimality ratio k is called *k-optimal.* An algorithm with the worst-case optimality ratio k is called a *k-approximation algorithm.*

If the processors are identical, then a linear time list scheduling algorithm which works with arbitrary precedence relations, gives a worst-case ratio $2 - \frac{1}{m}$ ([3]). An $O(n \log n)$ MULTIFIT algorithm gives the optimality ratio for identical processors 13/11 and for uniform processors $\leq 7/5$ (see [4], [5], [6]). Polynomial approximation schemes for uniform processors (the family of polynomial algorithms with optimality ratios arbitrary close to 1) were first suggested in [7].

With unrelated processors, a much weaker approximability results are known. The approximation scheme proposed in [8] is polynomial by n but non-polynomial by m. This algorithm with the optimality ratio $1 + \varepsilon$ has time complexity $O(n^{2m}/\varepsilon)$ and its space complexity is non-polynomial. For fixed m, i.e., when m is not an input on the problem, there is a liner by n polynomial approximation scheme by Jansen and Porkolab [17]. For non-fixed m, the first polynomial-time approximation algorithms for unrelated processors were proposed in [9] with the optimality ratio m. This result was essentially improved in [10] where polynomial-time algorithms with optimality ratio within $2\sqrt{m}$ were proposed. Breakthrough in the area was due to [1] in which a polynomial algorithm with optimality ratio 2 was proposed. It was also proved that there can exist no polynomial algorithm with optimality ratio 3/2 or less, unless $P = NP$. This work was preceded by the paper [11], in which first was brought into the play the linear programming for this problem and produced an efficient but still non-polynomial by m algorithm with optimality ratio 2. For a more detailed survey of the approximability results see [18].

In this paper, relying on the results from [1], we present an improved polynomial algorithm for unrelated processors with the Graham's performance bound for identical processors, i.e., with the worst-case ratio $2 - \frac{1}{m}$. This is the best result so far for $m > 2$. For $m = 2$, a linear time algorithm from [11] gives the similar result.

An *absolute error* estimates the quality of a schedule in absolute terms and is the difference between the makespan of this schedule and an optimal one. The *rounding theorem* from [1] provides with polynomial algorithms which construct schedules with an absolute error equal to the maximal job processing time p_{max}. We improve this result as well presenting a polynomial algorithm with an absolute error $\frac{m-1}{m} p_{\max}$. A similar result for identical processors was obtained in [12] in Theorem 1.2.

2 Preliminaries

In this section, we introduce the basic concepts and notations. A *schedule* assigns each task a processor, and also starting time on that processor, while a *distribution* deals only with the assignment of tasks to processors, but doesn't care about starting times of tasks on the assigned processors. In fact, the literature contains a number of results on distributions, we will mention some of them. Explicitly this concept has appeared for non-preemptive case under different names. For

example, in [12] it is used the term partition, and in [10] it is used assignment. For preemptive case we will use the term distribution which seems to us more adequate.

For job J and processor M, let us denote by $M(J)$ the time, which takes the complete execution of J on M, and by $M_\sigma(J)$ we denote the time, during which J is processed by M in the schedule σ. $\frac{M_\sigma(J)}{M(J)}$ is the part of J scheduled in σ on M (if preemptions are allowed, then this ratio can be any real number from the interval [0,1], and without preemptions we can have only 0s and 1s).

Every (preemptive or non-preemptive) schedule σ defines its corresponding distribution. The distribution δ_σ associated with schedule σ is a function which assigns to each pair J, M the value $\delta_\sigma(J, M) = \frac{M_\sigma(J)}{M(J)} \leq 1$. For every job J, scheduled in σ, we have $\sum\limits_M \delta_\sigma(J, M) = 1$, where the sum is taken over all machines in σ.

The concept of distribution may be introduced and investigated independently of the concept of schedule. This concept is simpler. We introduce more formal definitions. For the convenience, let us assume that all possible *jobs* constitute an universal set of jobs which we denote by $JOBS$. This set might be finite or infinite, but all schedules and distributions are defined only on finite subsets of $JOBS$. The *processors* or *machines* are defined as functions from $JOBS$ to nonnegative real numbers R^+. If M is a processor and J is a job, $M(J)$ is the time needed to execute J on M. The set of all processors is denoted by $PROC$. A *multiprocessor* or a *processor system* is a finite linearly ordered set of processors denoted by $\mathcal{M} = \{M_1, M_2, M_3, \dots M_m\}$. A multiprocessor consisting of m processors is called *m-multiprocessor*. A *job system* is a linearly ordered finite set of jobs $\mathcal{J} = \{J_1, J_2, \dots J_n\}$. A job system with n jobs is called *n-job system*.

A *distribution* is a function $\delta\colon JOBS \times PROC \to R^+$, such that for every job J, $\sum\limits_{M \in \mathcal{M}} \delta(J, M)$ is 0 or 1. Again, $\delta(J, M)$ is the part of job J assigned to machine M. The jobs, for which this sum takes value 1 are called *distributed* in δ or δ-*distributed*, and the rest of the jobs are called *non-distributed*. The set of all δ-distributed jobs is denoted by $JOBS(\delta)$. If $\delta(J, M) > 0$ we will say that job J is δ-*distributed* on the machine M (or the machine M is δ-*occupied* by J). We denote by $\delta(J)$ the set of all machines on which job J in δ is distributed. The set of all δ-occupied machines is denoted by $PROC(\delta)$. We will say that a *distribution* δ *distributes a job system* $\mathcal{J}$ *on a multiprocessor* $\mathcal{M}$ if $\mathcal{J} = JOBS(\delta)$ and $PROC(\delta) \subset \mathcal{M}$.

Let us note that a convex combination of two distributions, (i.e. $p\delta + (1 - p)\delta'$, $1 \geq p \geq 0$) is a distribution. The sum of two distributions δ and δ' is a distribution iff $JOBS(\delta) \cap JOBS(\delta') = \emptyset$. We will call such distributions *disjoint*. If inequality $\delta(J, M) \leq \delta'(J, M)$ holds for all J, M, then we will write $\delta \subset \delta'$ and say that δ is a *sub-distribution* of δ', and δ' is an *extension* of distribution δ. In this case, as it easy to see, $JOBS(\delta) \subset JOBS(\delta')$ and $\delta(J, M) = \delta'(J, M)$ for any M and δ-distributed job J.

$M_\delta(J) = \delta(J, M)M(J)$ is the time needed for machine M to execute the assigned to it in δ part of job J. $\sum\limits_{J\in\mathcal{J}} M_\delta(J)$ is called the *load time* of machine M in distribution δ and is denoted by $|\delta|_M$. The load time of machine $|\sigma|_M$ in a schedule σ is its load time in the associated distribution, so $|\sigma|_M = \sum\limits_{J\in\mathcal{J}} M_\sigma(J)$. Let us call the maximal load time of a machines in a distribution δ its *makespan* and denote it by $|\delta|_{\max}$. The makespan of a schedule σ will be also denoted by $|\sigma|_{\max}$.

The distribution δ is *optimal* if it has the minimal makespan among all distributions which distribute the same set of jobs on the same multiprocessor. The problem of constructing of an optimal distribution is equivalent to the following linear programming problem:

Minimize D_{opt}

$$\sum_{i=1}^{n} x_{i,j}t_{i,j} \le D_{\text{opt}}, \quad \sum_{j=1}^{m} x_{i,j} = 1, \quad i = 1, ..., n \quad j = 1, \dots m, \quad x_{i,j} \ge 0$$

To see this equivalence, we let $t_{i,j} = M_j(J_i)$, $x_{i,j} = \delta(J_i, M_j)$. For further references, we abbreviate this linear programming problem by $LP(D_{\text{opt}})$.

Construction of an optimal schedule can be split into two stages. On the first stage we construct an optimal distribution, and on the second stage we construct an optimal schedule with this distribution. If the distribution is non-preemptive, the second stage is trivial. The problem of construction of an optimal schedule, associated with the given distribution, was thoroughly investigated in [13]. The concept of an *open shop*, introduced in this paper, is almost the same as that of a distribution. Let δ be a distribution of $\mathcal{J}$ on $\mathcal{M}$. Then one can interpret every $J \in \mathcal{J}$ as a job consisting of m subtasks, where task number i has to be performed on processor M_i and its execution takes the time $(M_i)_\delta(J)$. In this way every distribution generate an open shop and vice-versa.

Unfortunately, not every preemptive distribution has an associated schedule with the same makespan. The *processing time* $|\delta|^J$ of a job J in a distribution δ is defined as the total time during which this job is processed on all machines, i.e. $|\delta|^J = \sum\limits_{M} M_\delta(J)$. Let us denote by $|\delta|^{\max}$ the maximum of $|\delta|^J$. So $|\delta|^{\max}$ is the largest processing time in δ. Since each job has to be performed *sequentially*, i.e., a job cannot be processed at any moment by more than one machine, the makespan of every schedule σ associated with a given distribution δ cannot be less than $|\delta|^{\max}$. Let us call the *sequential makespan* of a distribution the maximum between $|\delta|_{\max}$ and $|\delta|^{\max}$. We see that the makespan of every schedule associated with a given distribution cannot exceed its sequential makespan. Now, a principal result from [13] can be formulated as follows.

Theorem Gonzales and Sahni [13]) *There exists a polynomial algorithm which constructs for every distribution an associated schedule with makespan equal to the sequential makespan of this distribution.*

Based on this theorem, in [14] the problem of constructing of an optimal preemptive schedule is reduced to the following linear programming problem $LP(S_{\text{opt}})$:

Minimize S_{opt},

$$\sum_{i=1}^{n} x_{i,j}t_{i,j} \le S_{\text{opt}} \quad \sum_{j=1}^{m} x_{i,j}t_{i,j} \le S_{\text{opt}},\ x_{i,j} \ge 0,$$

$$\sum_{i=1}^{n} x_{i,j} = 1, \quad i = 1, ..., n, \quad j = 1, ..., m$$

Any solution of this problem gives a distribution with the minimal sequential makespan.

3 Previous Results on Rounding

Thus the construction of an optimal preemptive distribution is polynomially solvable in opposite to the non-preemptive case, which is a subject of our study. Let us first give the *rounding approach*, first applied in [11] and later essentially improved in [1]. This approach allows us to produce good approximation to an optimal non-preemptive schedule.

For a real x, let $[x]$ and $\{x\}$ be its integral and fractional parts, respectively (we note that we use $\{.\}$ for representation of sets as well). So $x = [x] + \{x\}$, $[x]$ is integer and $0 \le \{x\} < 1$. $[\delta(J, M)]$ and $\{\delta(J, M)\}$ define distributions $[\delta]$ and $\{\delta\}$, which we will call the *integral* and the *fractional* parts, of δ, respectively; clearly, $[\delta]$ and $\{\delta\}$ are disjoint. A distribution is *integral* or *non-preemptive* if its fractional part is 0, and in this case it coincides with its integral part. Jobs distributed by $\{\delta\}$ are said to be *preempted* in δ. An integral distribution δ is a *rounding* of another distribution δ' if it distributes the same jobs and $\delta = [\delta']$. So $\delta' = \delta + \delta_0$, where $\delta_0 = \{\delta'\}$.

To find a non-preemptive distribution, close to an optimal one, the rounding method looks for an optimal *extremal* preemptive distribution and *rounds* it. A distribution δ is *extremal* if it cannot be represented in the form $\frac{1}{2}(\delta' + \delta'')$, where δ' and δ'' are different distributions, such that for every machine M, $|\delta'|_M = |\delta''|_M = |\delta|_M$. The importance of extremal distributions shows the following

Extremality Principle. *All distributions constructed by linear programming solution of $LP(D_{\text{opt}})$ are extremal.*

Proof. Denote by Δ the set of all distributions of an n-job system $\mathcal{J}$ on an m-multiprocessor $\mathcal{M}$. This set represents a convex (possibly unbounded) polytope in space R^{nm}. In $LP(D_{\text{opt}})$, consider a subset Δ' of product $R^{mn} \times R$ defined by conditions $\Delta' = \{(\delta, D) \mid \delta \in \Delta, D \in R, |\delta|_{\max} \le D\}$.

If $\delta = \frac{1}{2}(\delta_1 + \delta_2)$ where $|\delta|_M = |\delta_1|_M = |\delta_2|_M$ for all machines M, then $(\delta, D) \in \Delta'$ implies $(\delta_i, D) \in \Delta'$ for $i = 1, 2$. Hence, (δ, D) is a middle point

of (δ_1, D) and (δ_2, D), and therefore is not a vertex of Δ'. But the linear programming works only with vertices of Δ'. Hence they work only with extremal distributions.

The rounding method is based on the following known principle (see [15], [11], [1]), which will be proved the next section.

The Preemption Bounding Principle. *If δ is an extremal distribution on a multiprocessor $\mathcal{M} = \{M_1 \dots M_m\}$, then the number of jobs in $JOBS(\{\delta\})$ (i.e., preempted jobs in δ) is less than m.*

Let us remark that for an integral distribution δ, $\delta(J)$ represents a unique machine. Let us say that an integral distribution δ is $1-1$ if $\delta(J) \neq \delta(J')$ for every δ-distributed jobs J and J'. Since the number of preempted jobs in an extremal distribution does not exceed the number of machines (see the Preemption Bounding Principle), we can accomplish a $1-1$ *rounding*, i.e., to find a distribution δ', for which $\delta' - \delta$ is a $1-1$ distribution. In particular, applying $1-1$ rounding to an optimal extremal distribution, we immediately obtain the following result (p_{max} below is the maximal task processing time):

A 1-job Approximation Theorem. *It is possible to construct a distribution with the makespan, exceeding the optimal makespan by no more than $p_{\max}$ in polynomial time.*

As we will see below, even an optimal $1-1$ rounding can be accomplished in polynomial time. Let us define the *selection problem* as follows. We say that a given family of subsets $\{X_i\}_{i=1,\dots,k}$ of a set X is *selectable* if there exist such sequence of point $x_1, \dots x_k \in X$ (called *selection*), that $x_i \in X_i$ for all $i \leq k$ and all x_i are distinct.

This selection problem can be easily solved via the *complete matching* problem in a bipartite graph $\{V_1, V_2, E\}$, with vertices $V_1 = X$ and $V_2 = \{1, 2, \dots k\}$, where pair x, i is an edge iff $x \in X_i$. This matching problem is known to be polynomially solvable via the maximal flow algorithm.

Due to the Hall's Marriage Theorem (see, for example [16]), our selection problem has no solution (i.e., there is no complete matching) iff there exists a subset $Y \subset X$ with the number of elements, less than the number of elements in the specially defined subset of V_2. In particular, this subset contains an element $l \in V_2$ iff there exists $y \in Y$, such that $y \in X_l$.

Lemma 1. *Let $\mathcal{J}$ be an n-job system, $\mathcal{M}$ be an m-multiprocessor, $m \geq n$ and $\{c_i\}_{i \leq n}$ be real numbers. Then it is possible to construct in polynomial time a $1-1$ distribution δ of $\mathcal{J}$ on $\mathcal{M}$, such that $|\delta|_{M_i} \leq c_i$ for all i, or to prove that such distribution does not exist.*

Proof. For every job J let $\mathcal{M}(J) = \{M \in \mathcal{M} \mid M_i(J) \leq c_i\}$. The constriction of $1-1$ distribution with $|\delta|_{M_i} \leq c_i$ is equivalent to the selection problem for the family $\mathcal{M}(J)$.

This lemma with lemma 1 in [1] gives us the following:

Theorem 1. *Let $\mathcal{J}$ be an n-job system, and $\mathcal{M}$ be an m-multiprocessor, such that $n \leq m$. Then the optimal $1-1$ -rounding can be constructed in polynomial time.*

4 Acyclicity of Distributions

In [1] it is considered an optimization problem for the distributions with the additional restrictions, which forbid some jobs to be distributed on some machines. The linear programming problem corresponding to this *restricted optimization problem* can be obtained from $LP(D_{\text{opt}})$ if some inequalities of the type $x_{ij} \geq 0$ are changed to the equalities $x_{ij} = 0$ ($x_{ij} = 0$ eliminates the possibility of distribution of any part of ith job to jth machine). The Extremality Principle from Section 3 also holds for this problem; the proof of this fact is similar to that of Section 3.

To specify a restricted optimization problem, we assign to each job J the set of machines $\mathcal{M}_J$, the ones, on which it is allowed to distribute J. We shall call such assignment a *job-machine configuration*. We will use $\mathcal{C}$ to denote the configurations. Formally, a job-machine configuration $\mathcal{C}$ is a multi-valued mapping $\mathcal{C}\colon JOBS \to PROC$. The set of machines, on which job J is distributed in $\mathcal{C}$ is denoted by $\mathcal{C}(J)$. We will consider only *finite* configurations; for some Js, $\mathcal{C}(J)$ might be empty.

A distribution δ is called *restricted* by configuration $\mathcal{C}$ or $\mathcal{C}$*-restricted* if $\delta(J) \subset \mathcal{C}(J)$ for all $J \in JOBS$. The set of $\mathcal{C}$-restricted distributions is convex. A distribution of a job system $\mathcal{J}$ with the minimal makespan, among all $\mathcal{C}$-restricted distributions

For a distribution δ, in [1] the so called *configuration graph* $G(\delta)$ is introduced. $G(\delta)$ is a bipartite graph $\{V_1, V_2, E\}$, such that $V_1 = JOBS(\delta)$, $V_2 = PROC(\delta)$ and there is an edge, corresponding to the pair J, M in $G(\delta)$ iff $\delta(J, M) > 0$. We will call a distribution *connected* if its configuration graph is connected.

A sub-distribution δ' of a distribution δ is called its *component* if $G(\delta')$ is component of connectedness of $G(\delta)$. From the definitions immediately follows

Lemma 2. *For different components δ_i and δ_j the respective sets of occupied machines are disjoint, i.e., $PROC(\delta_i) \cap PROC(\delta_j) = \emptyset$.*

Lemma 3. *Let δ, δ' and δ'' be such distributions that $PROC(\delta) \cap PROC(\delta') = PROC(\delta) \cap PROC(\delta'') = \emptyset$ and $|\delta'|_M = |\delta''|_M$ for all machines. Then $|\delta' + \delta|_M = |\delta'' + \delta|_M$ for all M.*

Lemma 4. *If $\delta = \frac{1}{2}(\delta' + \delta'')$ then $\delta(J) = \delta'(J) \cup \delta''(J)$ for all J.*

The proofs are left to the reader.

Lemma 5. *A distribution is extremal iff all its components are extremal.*

Proof. Suppose δ has a non-extremal component δ_0 such that $\delta_0 = \frac{1}{2}(\delta_0' + \delta_0'')$ and $|\delta_0|_M = |\delta_0'|_M = |\delta_0''|_M$. Then processors occupied in δ_0' and δ_0'' are included in $PROC(\delta_0)$. Let $\delta_1 = \delta - \delta_0$. Then $PROC(\delta_0) \cap PROC(\delta_1) = \emptyset$. Further, let $\delta' = \delta_0' + \delta_1$ and $\delta'' = \delta_0'' + \delta_1$. Then $\delta = \frac{1}{2}(\delta' + \delta'')$ and $|\delta'|_M = |\delta''|_M = |\delta|_M$ for all M. Now the load times are equal because of lemma 3.

In opposite direction, suppose δ is not extremal and consider its representation $\delta = \frac{1}{2}(\delta' + \delta'')$ where $|\delta|_M = |\delta'|_M = |\delta''|_M$ for all M. Let J be a job for which $\delta'(J) \neq \delta''(J)$ and δ_0 be a component of δ for which $J \in JOBS(\delta_0)$. Denote by δ'_0 and δ''_0 restrictions of δ' and δ'' respectively, on $JOBS(\delta_0)$. Then $\delta_0 = \frac{1}{2}(\delta'_0 + \delta''_0)$ and to prove the non-extremality of δ_0 it is sufficient to check equality of the load times. Let $M \in PROC(\delta_0)$. As $\delta'(J) \subset \delta(J)$ for all $J \notin JOBS(\delta_0)$, we obtain $M \notin \delta'(J)$. Hence, all jobs distributed by δ' on M belong to $JOBS(\delta_0)$ and are distributed by δ'_0. Therefore, $|\delta'_0|_M = |\delta'|_M = |\delta_0|_M$. If $M \notin PROC(\delta_0)$, then for all $J \in JOBS(\delta_0)$ $M \notin \delta(J)$ and hence $M \notin \delta'(J)$. This implies that the load time of M in δ'_0, as well as in δ_0, is 0. Thus we proved equality of the load times for δ'_0. We use the similar reasoning for δ''_0 and the lemma is proved.

Let us say that a distribution δ is *synchronous* over multiprocessor $\mathcal{M}$ if $|\delta|_M = |\delta|_{M'}$ for all $M, M' \in \mathcal{M}$.

Lemma 6. *A connected $\mathcal{C}$-optimal distribution δ is synchronous over $PROC(\mathcal{C})$.*

Proof. Suppose δ is not synchronous and let M be a machine for which $|\delta|_{\max} > |\delta|_M$. Consider a machine M' with the maximal load time. Since $G(\delta)$ is connected, there exists a path in it connecting M and M'. Let $M = M_1, J_1, M_2, J_2, \ldots, M_{k+1} = M'$ be such a path and let i be the maximal index, such that $|\delta|_{M_i} < |\delta|_{\max}$. J_i occupies both M_i and M_{i+1}. Let us choose $\varepsilon > 0$ so small that $\varepsilon < \delta(J_i, M_{i+1})$ and $\varepsilon M_i(J_i) + \delta(J_i, M_i) < 1$, and define a new distribution δ' as follows. $\delta'(J_i, M_i) = \delta(J_i, M_i) + \varepsilon$, if $\delta'(J_i, M_{i+1}) = \delta(J_i, M_i) - \varepsilon$ and $\delta'(J, M) = \delta(J, M)$ for any other job-machine pair.

The obtained distribution δ' contains less machines with the load time $|\delta|_{\max}$, and has the same configuration graph as δ. Repeating the above procedure, we can construct a distribution with the same configuration as δ and with a smaller makespan. But this contradicts the optimality of δ. The lemma is proved.

The number of machines in $\delta(J)$ minus 1 will be called the *number of preemptions* of J in δ and will be denoted by $\pi_\delta(J)$. $\pi(\delta) = \sum_{J \in \mathcal{J}} \pi_\delta(J)$ is the total number of preemptions in δ.

Lemma 7. *The total number of preemptions in every connected $\mathcal{C}$-optimal extremal distribution δ is strictly less than the number occupied machines in δ.*

Proof. Let δ occupy machines $M_1, \ldots M_m$ and let $J_1, J_2, \ldots J_k$ be the preempted jobs in δ. For every $i \leq k$, let $j(i)$ be the first j for which $M_j(J_i)$ is fractional. Denote by P the set of pairs i, j, $j \neq j(i)$ and such that $0 < M_i(J_j) < 1$. The number of elements in P is $\pi(\delta)$. Let $\varepsilon = \min\{M_i(J_j)\}_{(i,j) \in P}$. For every real function f on P, such that $|f(i,j)| \leq \varepsilon/m$, let $\delta_f(J_i, M_{j(i)}) = \delta(J_i, M_{j(i)}) - \sum_{j|(i,j) \in P} f(i,j)$, $\delta_f(J_i, M_j) = \delta(J_i, M_j) + f(i,j)$ if $(i,j) \in P$, and let $\delta_f(J, M) = \delta(J, M)$ otherwise. The distribution δ_f, as it follows from its definition, is $\mathcal{C}$-restricted. To define a linear mapping l of a ε/m-cube of Euclidean space Q of dimension $\pi(\delta)$ into $(m-1)$ dimensional space, enumerate pairs in P. Then each point $x \in Q$ corresponds to a real function f_x on P, and we can define

$l(x)$ as a vector in which ith component is $|\delta_{f_x}|_{M_i} - |\delta|_{M_i}$. If $\pi(\delta) \geq m$, then the kernel of the mapping l is nontrivial. Let $y \in Q$ be a nonzero point for which $l(x) = 0$ and let f be the function corresponding to x. If $|\delta_f|_{M_m} = |\delta|_{M_m}$, then $|\delta_{-f}|_{M_m} = |\delta|_{M_m}$ and we come to a contradiction with the extremality of δ, because $\delta = \frac{1}{2}(\delta_{-f} + \delta_f)$. The optimality of δ implies that $|\delta_f|_{M_m} > |\delta|_{M_m}$. Indeed, if $|\delta_f|_{M_m} < |\delta|_{M_m}$, then δ_f is as well optimal, but not synchronous. But the same reasons applied to δ_{-f} gives us the similar inequality $|\delta_{-f}|_{M_m} > |\delta|_{M_m}$. Then we come to a contradiction with $\delta = \frac{1}{2}(\delta_{-f} + \delta_f)$. The lemma is proved.

We will call distribution *acyclic* if its configuration graph is acyclic. Let us say that a distribution δ is *componentwise $\mathcal{C}$-optimal* if each its component is $\mathcal{C}$-optimal. Our main result now can be formulated as follows.

Theorem 2. *Every componentwise optimal and extremal distribution is acyclic.*

Proof. If a distribution is connected, then the number of its preemptions is less than the number of the corresponding occupied machines (by lemma 7). Hence its configuration graph has less edges than vertices. For a connected graph this implies the acyclicity. If the distribution is not connected, then consider its components. They are extremal by lemma 5. They are as well $\mathcal{C}$-optimal by our assumption. Hence the same argument shows their acyclicity.

5 Consolidation of Distributions

Let us say that a non-preemptive (integral) distribution δ is a *consolidation* of a preemptive distribution δ', if it has the same domain and $\delta(J, M) = \delta'(J, M)$, for all J which are not preempted in δ'.

The main result of this section is the following theorem

Theorem 3. *For every acyclic distribution δ on an m-processor, it is possible to construct in polynomial time a consolidation δ', such that* $|\delta'|_{\max} \leq |\delta|_{\max} + \frac{m-1}{m} p^{\delta}_{\max}$

The proof is based on some delicate considerations connected with graphs. Besides the introduced earlier configuration graph from [1], we shall consider another type of graph for presenting the preemptive structure of the distributions. We call it a *preemption graph* and denote it by G_δ. G_δ has less nodes and edges than the corresponding configuration graph.

The nodes of G_δ represent the machines, and edges represent the jobs. There is an edge in joining a pair of nodes in G_δ, if the job which represents this edge is shared by the machines which represent these nodes. The preemption graph, in general, is a multi-graph. But if the configuration graph is acyclic, then the corresponding preemption graph is simple. Indeed, it is sufficient to prove that $\delta(J_1) \cap \delta(J_2)$ cannot contain two machines. But if it contains two machines M_1 and M_2, then we will have a nontrivial cycle M_1, J_1, M_2, J_2 in $G(\delta)$.

Preemption graph may have cycles even if configuration graph is acyclic. For example, if $\delta(J)$ contains three machines M_1, M_2 and M_3, then the vertices corresponding to M_i, $i = 1, 2, 3$, form a cycle in G_δ.

A *reduced preemption graph* G'_δ has less edges than G_δ has and this graph is acyclic iff $G(\delta)$ is acyclic. The structure of G'_δ (unlike that of G_δ and $G(\delta)$) depends not only on the distribution δ, but also on the order, in which the machines are numbered. G'_δ is a subgraph of G_δ, obtained by deleting the so called *redundant edges* in G_δ. Again, whether an edge is redundant or not, depends on the order in $\mathcal{M}$. An edge $(x, y) \in G_\delta$ is redundant, if there exist two (or more) intermediated edges (x, z) and (z, y), such that the index of machine z is more than that of machine x and less than that of machine y, and all x, y and z share the same job.

Proposition 1. *G'_δ is acyclic iff $G(\delta)$ is acyclic. The number of edges in G'_δ is equal to the number of preemptions in the distribution δ.*

Lemma 8. *Let $P_1, P_2, \ldots, P_k$ be connected subgraphs of an acyclic graph G, having in common at most one node. Then for some i, the intersection of P_i with $\cup P_j, j = 1, 2, ..., k, j \neq i$, is a single node.*

Proof. Let $N_1, \ldots N_k$ be all nodes of G, which are common for some pairs of our connected subgraphs, and let G' be the minimal connected subgraph of G containing all N_is. The acyclicity of G implies the acyclicity of G'. Besides, G' does not have any single degree node, different from some N_i. Indeed, if N were a single degree node in G' different from any N_i then we would reduce G' by eliminating N and the corresponding edge.

Let N be a single-degree node of G' and e be the edge in G', corresponding to N. As $N = P_i \cap P_j$ for some i, j, either P_i or P_j, suppose P_i, does not contain e. Suppose that P_i contains a node $N_j \neq N$. In this case we will have two different paths between N_j and N in G. The first such path is contained in P_i and does not pass through e, and the second one is in G' and passes through e. But then we would have a cycle in G which is a contradiction. Therefore, the intersection of P_i with the union of the rest of our connected subgraphs is exactly N and the lemma is proved.

A *weight function* on the graph G is a function w which assigns to each node N of G a nonnegative number $w(N)$, and such that the sum of all $w(N)$ is equal to 1. Let us say that a weight function w is *supported* by a subgraph P of G if it takes value 0 for all nodes in $G \backslash P$.

Lemma 9. *Let G be an acyclic graph with m nodes, $P_1, P_2, \ldots, P_k$ its covering by connected subgraphs which intersect in more than in one node. Further, let $w_1, w_2, \ldots, w_k, k < m$ be a weighted function on G such that w_i is supported by P_i. Then it is possible in polynomial time to construct a sequence of nodes $s_1, \ldots, s_k$, such that $s_i \in P_i$ for all i, and $\sum_{s_i = N} (1 - w_i(N)) \leq 1 - \frac{1}{m}$*

Proof. Applying the above lemma we can first order subgraphs $P_1, P_2, \ldots P_k$ of our decomposition in such a way that for all $i < k$, P_i intersect the union $P^i = \cup_{j>i} P_j$ in exactly one node, which we denote by N_i.

In the process of construction, we will need to keep an additional information for every node. An array of nonnegative integers $v(N)$. The algorithm is the following.

Step 1. (initialization) $v(N) := 0$ for all N, $j := 0$ (counter of cycles)

Step 2. Search for a node M in P_i, different from N_i and such that

$$w_i(M) \geq v(M)/m + 1/m,$$

(case 1): if such a node is founded then $s_i := M$

(case 2): otherwise, $s_i = N_i$, $v(N_i) := v(N_i) + \sum_{M \in P_i \setminus N_i} (v(M) + 1)$

Step 3. if $j < k$ then j:=j+1; goto step 2; else STOP

To prove correctness of this algorithm, first note that $\sum_{M \in P^i} v(M)$ is no more than the number of nodes in complement to P^i, for all stages in our construction. Indeed, from P^{i-1} to P^i we increase $\sum_{M \in G'} v(M)$ exactly by the number of the deleted nodes from P_i, or we do not change it at all.

The algorithm works without stopping up to the last step. When we choose s_k there is no N_k. So we have to find an M satisfying inequality $w_k(M) \geq v(M)/m + 1/m$. Suppose that there is no such M. In this case, for all M we have $w_k(M) < v(M)/m + 1/m$. If we sum these inequalities, for all $M \in P_k$, we obtain $1 < \frac{1}{m} \sum(v(M)+1)$. But $\sum(v(M)+1)$ as already noted, does not exceed m and we came to a contradiction.

Let us note that if s_i is different from N_i, then $w_i(M) \geq v(M)/m + 1/m$ and this point cannot be chosen in the sequel. On the other hand, during the whole process, for all i we have that

$$\sum_{j:s_j=N} (1 - w_j(N)) \leq v(N)/m,$$

because $v(N)$ is increased only when $N = N_i$ is selected, for some P_j, and it is increased by $\sum_{M \in P_i \setminus N} (v(M) + 1)$. But the condition of selection of N_i is that $w_i(M) < v(M)/m + 1/m$ for all $M \in P_j$. And sums of these inequalities provide desired results. To finish our prove it is sufficient to note that $v(N) \leq m - 1$ for all N.

Proof of Theorem 3 Let G be a reduced preemption graph of fractional part of our distribution. This graph is subgraph of G'_δ and therefore is acyclic. Let $J_1, \ldots J_k$ be all jobs preempted in δ. For every job J_i denote by P_i the subgraph which edges correspond to J_i. For every node M of this subgraph, let $w_i(M) = \delta(J_i, M)$. Then we obtain decomposition $\{P_i\}$ of G and the system of weights. By lemma 9 we choose for each job J_i, a machine $M(i)$. Now we define $\delta'(J_i, M(i)) = 1$ for all i. This completely defines this consolidation. The increase of the load time on machine M in δ' compared with that in δ is equal to $\sum_{i:M(i)=M} (1 - \delta(J_i, M))M(J_i)$. As $M(J_i) \leq p^\delta_{\max}$ for all i, this sum does not exceed $(1 - \frac{1}{m})p^\delta_{\max}$. The theorem is proved.

6 The Worst-Case Bounds

As a simple consequence of the acyclicity and consolidation theorems, we build the approximation algorithms in this section.

Given a configuration $\mathcal{C}$ and job system $\mathcal{J}$, construct by the linear programming a $\mathcal{C}$-optimal distribution δ of $\mathcal{J}$. Decompose it into components $\delta = \sum \delta_i$. For all i, construct by linear programming extremal $\mathcal{C}$-optimal distributions δ'_i of $JOBS(\delta_i)$. Then the sum $\sum \delta'_i$ represents an extremal componentwise $\mathcal{C}$-optimal distribution of $\mathcal{J}$. This distribution will be acyclic owing to the Acyclicity Theorem. Now applying consolidation algorithm of Theorem 3, we obtain a distribution which properties are given in the next theorem. Denote by $p^{\mathcal{C}}_{\max}$ maximum of $M(J)$, where $M \in \mathcal{C}(J)$, and denote by $D^{\mathcal{C}}_{\text{opt}}$ the makespan of $\mathcal{C}$-optimal distribution of $\mathcal{J}$.

Theorem 4. *For every job system $\mathcal{J}$ and every configuration $\mathcal{C}$ it is possible to construct in polynomial time an integral distribution δ, such that*

$$|\delta|_{\max} \leq D^{\mathcal{C}}_{\text{opt}} + \frac{m-1}{m} p^{\mathcal{C}}_{\max}$$

If we consider a *constant configuration*, i.e., a configuration $\mathcal{C}$, such that $\mathcal{C}(J) = \mathcal{M}$ for all $J \in \mathcal{J}$, then, from the above theorem, we immediately obtain the following improved version of the 1-job approximation theorem.

Corollary 1. *There is a polynomial algorithm which constructs an integral distribution of a job system $\mathcal{J}$ on a multiprocessor $\mathcal{M}$ with a makespan, exceeding the optimal one by no more than $\frac{m-1}{m} p_{\max}$.*

Now we find it useful to recall some basic ideas behind the earlier mentioned polynomial 2-approximation algorithm from reference [1]. One of the useful concepts, introduced in [1] was what we call the *balanced distribution*, a distribution, in which the jobs cannot be distributed on the machines, on which their execution time is "sufficiently large". More precisely, for a distribution δ, let us denote by $p^{\delta}_{\max}$ the maximum of $\{M(J) \mid \delta(J, M) > 0\}$ and call it the *δ-largest processing time*. Let us say that for some $B \in R^+$ δ is B-balanced, if $|\delta|_{\max} \leq B$ and $p^{\delta}_{\max} \leq B$. Denote by B_{opt} the minimum B for which there exist a B-balanced distribution. Note that $D_{\text{opt}} \leq B_{\text{opt}} \leq D^{\text{opt}}$. Indeed, every non-preemptive distribution δ is *auto-balanced* (that is $|\delta|_{\max}$-balanced) and this implies that $B_{\text{opt}} \leq D^{\text{opt}}$. We call an *optimally balanced* distribution a distribution which is B_{opt}-balanced. B_{opt} can be found in polynomial time as it is shown in [1].

In the polynomial 2-approximation algorithm from [1], first an optimally balanced extremal distribution δ is constructed and then *consolidated*. An integral distribution δ is a *consolidation* of a distribution δ', if δ and δ' distribute the same job system $\mathcal{J}$ on the same multiprocessor $\mathcal{M}$ and $\delta(J, M) = \delta'(J, M)$ provided by integrality of $\delta(J, M)$. Consolidation is a special sort of rounding. To build a consolidation from a given distribution, for each preempted job in this distribution a single machine is determined and the job is completely placed

(scheduled) on that machine. If δ' is a consolidation of a B-balanced distribution δ, then $|\delta'|_{\max} \leq |\delta|_{\max} + B$. Hence, a consolidation of an optimally balanced distribution has the makespan $\leq 2B_{\text{opt}} \leq 2D^{\text{opt}}$.

An ingenious trick is developed in [1] to consolidate an B-balanced distribution obtained by linear programming. This consolidation is produced by a matching in the corresponding configuration graph and is $1-1$. That is different preempted jobs occupy in consolidation different machines. This trick is based on the analysis of structure of the consolidation graph of the above distribution. It is proved that this graph is a *pseudo-forest* i.e., a graph with its all components having the edges no more than the nodes.

Let us denote by B^{opt} the minimal possible makespan of auto-balanced distributions. Auto-balanced distribution with such makespan call optimal auto-balanced. This distribution is extremal and hence acyclic. That is for such distributions the configuration graph is a forest. As easy to see $B_{\text{opt}} \leq B^{\text{opt}} \leq D^{\text{opt}}$. The same argument as presented in [1] show that B^{opt} is calculable in polynomial time. This B^{opt} represents a best known polynomially calculable lower estimation for D^{opt}.

The consolidation applied to the optimal auto-balanced distribution (which is not in general 1-1) gives a $2 - \frac{1}{m}$-approximation to optimum.

Theorem 5. *For every given configuration, a non-preemptive distribution with optimality ratio $2 - 1/m$ can be constructed in polynomial time.*

Given a configuration $\mathcal{C}$, construct by the linear programming a $\mathcal{C}$-optimal extremal distribution δ. This distribution is acyclic

Theorem 6. *There is a polynomial algorithm which constructs a schedule σ with makespan exceeding an optimal schedule by no more than $\frac{m-1}{m} p_{\max}$.*

References

1. J.K. Lenstra, D. B. Shmoys, E. Tardos "Approximation algorithms for scheduling unrelated parallel machines" *Mathematical programming*, 46, 259-271, 1990
2. Karp, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher, Eds., Plenum Press, New York,1972,pp.85-103
3. Graham R. L. "Bounds for certain multiprocessing anomalies" *Bell. Syst. Tech. J*, 45 (1966), 1563-1581
4. D. K. Friesen "Tighter bound for the MULTIFIT processor scheduling algorithm" *SIAM J. Comput.*, 13, n.1, Febr.1984,170-181.
5. Yue M. "On the exact upper bound for the multifit processors scheduling algorithm", *Ann. Oper. Res.*, 24 (1990), 233-259
6. D. K. Friesen, M. A. Langston "Bounds for MULTIFIT scheduling on uniform processors" *SIAM J. Comput.*, 12, n.1, Febr. 1983, 60-70.
7. D. S. Hochbaum and D. B. Shmoys "A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach", SIAM J. Comput., 17, n. 3 (1988), 539-551

8. Horowitz E. and Sahni S., "Exact and approximate algorithms for scheduling nonidentical processors", J. ACM, 23,n.5, (apr. 1976),317-327
9. Ibarra O.H., Kim C. E. "Heuristic algorithms for scheduling independent tasks on nonidentical processors",J. ACM, 24,2 (April 1977) 280-289
10. E. Davis, J. M. Jaffe *Algorithms for Scheduling Tasks on Unrelated Processors*, Journal of the ACM 28, 721-736 (1981).
11. C. N. Potts "Analysis of a linear programming heuristic for scheduling unrelated parallel machines" *Discrete Appl. Math.*, 10, 155-164 (1985)
12. G. Dobson, *Scheduling independent tasks on uniform processors*, SIAM J. Comput., Vol 13,No 4, November 1984,pp. 705–716
13. Gonzalez T. and S. Sahni, "Open Shop Scheduling to Minimize Finish time", *Journal of the ACM* 23, 665-679 (1976).
14. Lawler E.L. and J.Labetoulle, "On preemptive scheduling of unrelated parallel processors by linear programming", *J. of the ACM* 25, 612-619 (1978).
15. G. B. Dantzig " Linear programming and extensions" Princeton University Press, Princeton, NJ, 1963.
16. Berge C. "Graphs and Hypergraphs", American Elsevier, New York, 1973, p. 134
17. Jansen K. and L.Porkolab. "Improved approximation schemes for scheduling unrelated parallel machines". *Proc. STOC99*, 1999.
18. Hall L.A. "Approximation algorithms for scheduling". In *Approximation algorithms for NP-hard problems*, PWS Pub., 1997.

Fast Interpolation Using Kohonen Self-Organizing Neural Networks

Olivier Sarzeaud[1] and Yann Stéphan[2]

[1] ECTIA, 1 rue de la Noë, BP 92119, 44321 Nantes cedex 3, France,
Olivier.Sarzeaud@ectia.ec-nantes.fr,
http://www.ec-nantes.fr/ectia
[2] EPSHOM - CMO, 13 rue du Chatellier, BP 426,
29275 Brest cedex, France

Abstract. This paper proposes a new interpolation method based on Kohonen self-organizing networks. This method performs very well, combining an accuracy comparable with usual optimal methods (kriging) with a shorter computing time, and is especially efficient when a great amount of data is available. Under some hypothesis similar to those used for kriging, unbiasness and optimality of neural interpolation can be demonstrated. A real world problem is finally considered: building a map of surface-temperature climatology in the Mediterranean Sea. This example emphasizes the abilities of the method.

1 Introduction

Physical data interpolation is a common issue in Geosciences. For many variable of interest, the measurements are often sparse and irregularly distributed in time and space. Analyzing the data usually requires a numerical model, which samples the data on a regular grid. Mapping irregular measurements on a regular grid is done by interpolation, which aims to generalize, but not to create, information. A popular method to map geophysical data is kriging [1].

This method, based on the hypothesis that the measurements are realizations of a random variable, has been proven to be optimal under certain conditions. It requires to solve a system of linear equations at each point where the interpolation must be done, which might be computationally heavy.

This paper proposes an original interpolation method based on Kohonen self-organizing networks. The method is applied on the problem of building a surface-temperature climatology in the Mediterranean Sea. The method performs very well, combining an accuracy comparable with usual kriging methods with a much shorter computing time, and is especially efficient when a great amount of data is available.

The paper is organized as follows. Section 2 recalls the backgrounds of kriging techniques. Section 3 describes the adaptation of self-organizing maps to the spatial interpolation problem. The results of actual data interpolation in an oceanographic problem are presented and discussed. The last section draws conclusions and perspectives.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 126–139, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Optimal Interpolation

A model of a physical variable aims at predicting its value anywhere at any time. The simplest model is a numerical one, that is a discrete representation of the variable. To be efficient, this representation must be done under two constraints: on the one hand no information must be missed, on the other hand a reasonable amount of storage capacity is required. It must also be done on a regular grid, in order to be usable by most analyzes tools (plotting a map of the variable, computing Fourier Transform, ...).

2.1 Definition of Interpolation

Considering n values (obtained by measurements) of a variable Z_i at locations $x_i, 1 \leq i \leq n$, interpolation aims at building a numerical model of the variable on a regular pre-defined grid. A straightforward way to interpolate data on a specific location (a point of the grid) is to make a linear combination of the data:

$$Z^* = \sum_{i=1}^{n} \lambda_i Z_i \tag{1}$$

where Z^* is the estimated value. The problem is to compute the weights λ_i in order to minimize the estimation error. Practically, this is not feasible, because the true values are not known. It is thus necessary to make assumptions on the behavior of the variable to define the optimality.

The simplest methods give higher weights to the nearest data. The weights are somehow inversely proportional to the distance. This corresponds to an implicit assumption of continuity of the variable, which seems reasonable for physical variables. Anyway, it is possible to do better, taking into account the spatial correlation of the data. In this case, the weights are the solutions of a system of linear equations, that can be obtained by writing the minimization of the estimation error. This is kriging.

2.2 Kriging

Kriging is based on a statistical interpretation of the measures. Indeed, it assumes that the data are realizations of a random variable, that is: $Z_i = Z(x_i)$. Some hypothesis are required on the behavior of this random variable, usually that the expectation of an increment is null, and its variance only depends on the distance (intrinsic random variable [5]):

$$E[Z(x+h) - Z(x)] = 0 \tag{2}$$

$$Var[Z(x+h) - Z(x)] = C(h) \tag{3}$$

Therefore, on each point x_0 where the interpolation is to be done, it is possible to write analytically the expectation and variance of the estimation error $Z^*(x_0) - Z(x_0)$.

Unbiasness. The nullification of the expectation (ensuring that the estimation is not biased) leads to a constraint on the weights λ_i:

$$\sum_{i=1}^{n} \lambda_i = 1 \tag{4}$$

Optimality. The minimization of the variance (that is the optimality of the estimation) under the constraint of Eq. 4 leads to a system of linear equations, with coefficients depending on a model of the variance of the increment of the data [5]:

$$\begin{bmatrix} C_{11} & \dots & C_{1n} & 1 \\ \dots & \dots & \dots & \dots \\ C_{n1} & \dots & C_{nn} & 1 \\ 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \dots \\ \lambda_n \\ \mu \end{bmatrix} = \begin{bmatrix} C_{10} \\ \dots \\ C_{n0} \\ 1 \end{bmatrix} \tag{5}$$

where $C_{ij} = Var[Z(x_i) - Z(x_j)] = E[(Z(x_i) - Z(x_j))^2]$ and μ is a Lagrange multiplier.

Estimation Error. Once the weights are found, it is also possible to compute the (residual) variance of the estimation error at each point where an estimation is performed:

$$Var[Z^*(x_0) - Z(x_0)] = \frac{1}{2} \sum_{i=1}^{n} \lambda_i C_{i0} \tag{6}$$

This approach is also called objective analysis.

When a great amount of data is available, kriging at each point cannot be performed using all data, because it would lead to huge systems that may not be handled. Instead, it is necessary to choose a few data around the point where to interpolate. Furthermore, these data have to be chosen to avoid singularity of the system, which is usually done with the help of many geometric parameters in kriging products. Anyway, it remains that a system of linear equations must be solved on each point of the final grid, which is computationally heavy.

The main advantage of kriging is that it relies on strong theoretical backgrounds, which demonstrate that the interpolation is unbiased and optimal. The main drawbacks are:

- The hypothesis done on the random variable are strong. It is possible to relax them (allowing a determinist drift on the data for example), but the kriging system is then more complex. In any case, a model of variance of the increment of the variable must be computed, which can be very long when a lot of data is available.
- It is difficult to ensure that a system built with some data, even carefully chosen, will be regular. Therefore, especially with big data sets, it is possible to have wrong estimates. Very wrong estimates can usually be detected, because they simply are out of the range of the variable. Anyway, it remains

the possibility to have wrong estimates that are not far enough from the expected value to be detected.
– To make a numerical model on a grid, it is necessary to interpolate, that is to solve the system of equations, on each point of the grid. This again might be very long, depending on the desired resolution of the model.

3 Neural Interpolation

Kohonen networks are artificial neural networks. Some work has already been done and is presented elsewhere [8][9] on their use for adaptive meshing. It was shown that a simple modification of the basic Kohonen self-organizing algorithm (to constrain the peripheral neurons of the network to stay on the border of the domain) allows to produce valid meshing, with some advantages over classical methods. The use of Kohonen networks for neural interpolation also relies on a slight modification of the basic self-organizing algorithm.

3.1 Kohonen Self-Organizing Networks

In their widely used form, Kohonen networks consist of a matrix of neurons, each neuron being connected to its four nearest neighbors through fixed connexions (this form is called a map). All neurons are also excited by the same input, a vector of any dimension, through weighted connexions (figure 1). The role of the fixed connexions is to create a competition process between the neurons, so that the one whose weights are the closest to the current input produces the higher output. This competition is usually simulated by a simple computation of the distances between the input and all the neurons, and selection of the neuron with the smallest distance. This neuron is called the cluster of the network.

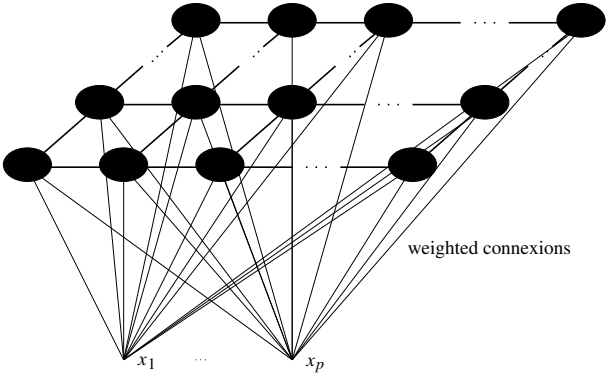


Fig. 1. Structure of a self-organizing map.

Kohonen has proposed a learning rule, that modifies the weights of the network, so as to produce interesting representations of the input space [4]. Indeed, at the end of the learning process:

- each neuron is sensitive to a particular zone of the input space;
- neighbor neurons are sensitive to near zones;
- the neurons distribution tends to approximate the probability density of the inputs presented during learning.

For these reasons, the usual representation of a self-organizing map is done by plotting the neurons, linked by their fixed connexions, using the weights as coordinates in the input space (see figure 2).

Learning Rule. Let:

- p be the dimension of the input space;
- $x(t) = (x_1(t), x_2(t), ..., x_p(t))$ be the input vector at time t;
- $w^k(t) = (w_1^k(t), w_2^k(t), ..., w_p^k(t))$ be the weight vector of neuron k at time t.

At each time step t, the cluster $c(t)$ of the network is searched:

$$\begin{aligned} c(t) &= c\left(\{w^k(t)\}, x(t)\right) \\ &= k / \|w^k(t) - x(t)\| \leq \|w^l(t) - x(t)\| \quad \forall l \end{aligned} \tag{7}$$

where the norm is usually the euclidian distance. The weights of the neurons are then adapted with the following rule:

$$w^k(t+1) = w^k(t) - \alpha(t) h(k, c(t))(w^k(t) - x(t)) \tag{8}$$

where $\alpha(t)$ is a time-decreasing gain factor, and $h(k, c(t))$ a neighboring function. This function depends on the topologic distance, measured on the map, between the neuron k and the cluster $c(t)$. It takes a maximum value of 1 if the distance is null (neuron k is the cluster), and decreases when the distance increases. The topologic distance between two neurons is the distance between their row and column indices in the map.

This algorithm is very robust, and numerous constraints can be applied to the neurons without changing its properties. For example:

- Initializing the network as a regular grid in the whole space considerably reduces the computation time.
- Constraining the peripheral neurons of the network to slide on the border of the domain allows the production of naturally adapted meshing [9]. The right part of figure 2 gives an illustration of such a meshing, produced with the same parameters as the left part, except the constraint.

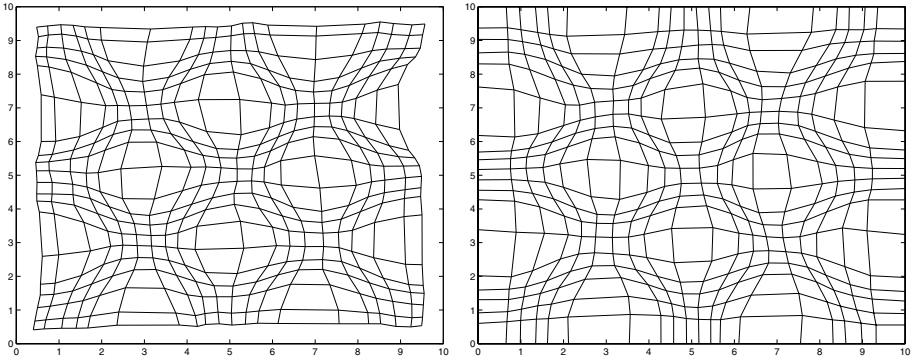


Fig. 2. Two maps organized using data chosen randomly on the black squares of a chess. The only difference between the two is that peripheral neurons are constrained to slide on the border of the domain on the right.

3.2 Neural Interpolation Algorithm

At each time step t, the input $x(t)$ of the network is a three-dimensional vector, the first two dimensions giving the location of the measure, and the third one its value. We will note this decomposition: $x(t) = (x_{loc}(t), x_{val}(t))$ ($loc = 1, 2$ and $val = 3$). Each neuron k thus has a three-dimensional weight vector $w^k(t)$. The only required modification of the basic self-organizing algorithm is that the selection of the cluster $c(t)$ be performed on the first two dimensions only:

$$\begin{aligned} c(t) &= c(w^k_{loc}(t), x_{loc}(t)) \\ &= k / \|w^k_{loc}(t) - x_{loc}(t)\| \leq \|w^l_{loc}(t) - x_{loc}(t)\| \quad \forall l \end{aligned} \tag{9}$$

This means that the cluster is chosen in the geographical space, according to the data location only, and is completely independent from the measure value. The idea is to trust the locations rather than the measures, allowing thus very different values measured on close points to be combined. Once the cluster is found, the weight modification applies on all three weights of the neurons, as presented in Eq. 8.

In this approach, the interpolation points cannot be chosen beforehand. Instead, they are determined during the learning process, and correspond to the final locations of the neurons. Therefore, at the end of the algorithm, we still do not have the values on a regular grid, and a post-processing is required. The question is thus: what is the advantage of the final irregular distribution of the neurons over the initial irregular distribution of the data? If the number of neurons is lower than the number of measures, complexity is reduced without loss of information. The neurons set is the best representation in space of the data set [6]. If the associated values are optimal, it is then possible to re-grid the values with a simple interpolation method, with no loss of precision.

At any time step t, it can easily be shown that the weight vector of a neuron k is a combination of the input vectors presented until then and of its initial value:

$$w^k(t) = \sum_{i=1}^{t} a^k(i) \prod_{j=i+1}^{t} (1 - a^k(j)) x(i) \quad + \quad \prod_{i=1}^{t} (1 - a^k(i)) w^k(0) \tag{10}$$

where $a^k(i) = \alpha(i)h(k, c(i))$. The initial value $w^k_{val}(0)$ can always be set to 0 without loss of generality. The weights of the combination do not depend on the third dimension (value) of the input and neuron weight vectors, but rather on the gain factor and on the locations of the input and neuron. Therefore, the third weight of the neuron is at any time a true *linear combination* of the data presented until then:

$$w^k_{val}(t) = \sum_{i=1}^{t} \lambda^{kt}(i) x_{val}(i) \quad with \quad \lambda^{kt}(i) = a^k(i) \prod_{j=i+1}^{t} (1 - a^k(j)) \tag{11}$$

Is this linear combination optimal? We will show this in the same way as for kriging. We first assume that the data are realizations of a random variable Y. But this time, this variable is not stationary, and is the sum of an intrinsic random variable (the Z variable used in kriging) and a determinist *linear* drift m:

$$Y(x) = Z(x) + m(x) \tag{12}$$

We can first make the following analogies with kriging notations:

$$\begin{array}{l} x_i = x_{loc}(i) \;\; Y(x_i) = x_{val}(i) \\ x_0 = w^k_{loc}(t) \; Y^*(x_0) = w^k_{val}(t) \\ \lambda_i = \lambda^{kt}(i) \end{array}$$

With these notations, the formula of Eq. 11 can be rewritten:

$$Y^*(x_0) = \sum_{i=1}^{t} \lambda_i Y(x_i) \tag{13}$$

If t is sufficiently big, the second term of Eq. 10 (influence of the initial weight vector of the neuron) can be neglected, therefore leading to:

$$x_0 = \sum_{i=1}^{t} \lambda_i x_i \tag{14}$$

Unbiasness. The expectation of the estimation error at point x_0 can be written:

$$E[Y^*(x_0) - Y(x_0)] = E\left[\sum_{i=1}^{t} \lambda_i Z(x_i) - Z(x_0)\right] + \sum_{i=1}^{t} \lambda_i m(x_i) - m(x_0) \tag{15}$$

This expression has a first probabilist term and a second determinist term. Each of them must be nullified to ensure unbiasness. The second term is naturally

null, because the drift is linear and the final location x_0 of the neuron is a linear combination of the location of the data presented (Eq. 14):

$$m(x_0) = m(\sum_{i=1}^{t} \lambda_i x_i) = \sum_{i=1}^{t} \lambda_i m(x_i) \tag{16}$$

We can thus say that *moving the neurons filters the drift.*

Nullity of the first term requires the same condition as for kriging: the sum of the λ_i must be 1. Let us note A_t this sum. Developping A_t using Eq. 11 leads to:

$$A_t = 1 - \prod_{i=1}^{t} (1 - a^k(i)) \tag{17}$$

A_t tends to 1 when t increases iff the *log* of the product tends to minus infinity. A first order development gives:

$$log(\prod_{i=1}^{t} (1 - a^k(i))) = -\sum_{i=1}^{t} a^k(i) + o(a^k(i)^2) \tag{18}$$

If the gain factor $\alpha(t)$ follows a decreasing law of the type $1/t^\beta$ with $0 \leq \beta \leq 1$, which is the usual convergence condition of Kohonen networks [7], then the sum goes to infinity, and A_t converges to 1.

Optimality. To show the optimality, that is the minimization of the variance of the estimation error, is much more difficult. Indeed, the neural interpolation algorithm never uses an explicit knowledge of the variance of the increment of the random variable. Therefore, an assumption is needed on how this variance is taken into account in the algorithm. We suppose first that the variance only depends on the distance between data points in the representation space of the map instead of the input space. Furthermore, we suppose that the neighboring function is a good representation of this variance:

$$C_{ij} = C_0(1 - h(c(i), c(j)) \tag{19}$$

where C_0 is a normalisation factor. This assumption is intuitively true for very big data sets. Indeed, in this case, measurements were made where variations were expected rather than where the variable was known to be stable. The resulting distribution thus reflects the variability of the measures.

The (determinist) drift naturally disapears when writing the variance of the estimation error. Optimality is therefore ensured under the same condition as for kriging (first n lines of the system of Eq. 5):

$$\sum_{i=1}^{t} \lambda_i C_{ij} = C_{j0} \quad \forall j \tag{20}$$

where $C_{j0} = C_0(1 - h(c(j), k))$, k being the considered neuron. Under the hypothesis of a unitary neighborhood, λ_i is non zero only when $c(i) = k$, thus when $C_{ij} = C_{j0}$. The sum of the λ_i being 1 as shown above, this demonstrates optimality.

Estimation Error. If a model of variance of the increments is available, the variance of the estimation error can be iteratively computed during the learning process. An updating rule similar to the one of Eq. 8:

$$C^k(t+1) = C^k(t) - \alpha(t)h(k, c(t))(C^k(t) - C_{t0}) \tag{21}$$

would lead to the following result:

$$C^k(t) = \sum_{i=1}^{t} \lambda_i C_{i0} \tag{22}$$

which is simply twice the variance of the estimation error defined in Eq. 6.

However, as no model of variance is required for neural interpolation, we would prefer not to have to compute one at all. This model is an analytic representation of the mean of the squared increments between data, function of their distance. Neural interpolation being a stochastic process, we propose to use at each time step the squared increment between the data presented and each neuron, instead of a model of what this value should be. This leads to the following new updating rule:

$$C^k(t+1) = C^k(t) - \alpha(t)h(k, c(t))(C^k(t) - (w^k_{val}(t) - x_{val}(t))^2) \tag{23}$$

This rule allows a better understanding of the local variability of the measures. Outliers can be more easily detected on the resulting map, because the local error they produce is not smoothed, as would be the case with a general model of variance. The error map rather reflects the variability of the data than their density.

4 Comparison

4.1 Algorithmic Complexity

It is important to compare the algorithmic complexity of kriging and neural interpolation, according to the numbers of data n, interpolation points m and iterations t.

Concerning kriging, two steps are required: computation of the model of variance first, and estimation itself. The first step needs a constant number of operations a for all n^2 couples of data points. The second step consists in solving a system of p linear equations with p unknowns, where p is the number of data points considered. Remember that when the data set is big, all data cannot be considered to build the system of equations. We do not take into account the time needed to cleverly choose these p points. The resolution needs $bp^3/10$ operations, where b is a constant depending on the chosen method, the same order as a. For m estimation points, kriging complexity is thus: $an^2 + bmp^3/10$.

Concerning neural interpolation, two steps are also required at each time step. First, the distance between the considered data point and all neurons is

computed (we consider that we have m neurons). The cluster can be found in constant time, under some simple hypothesis [10]. Then, all neurons are updated according to their distance to the cluster. These two steps require cmt operations, where c is the same order as a and b.

Neural interpolation and kriging have a comparable cost if the number of iterations t is the same order as the maximum of n^2/m and $p^3/10$. Clearly, when very few data points are available, kriging is faster, while when a lot of data points are available neural interpolation is faster. Two numerical examples are given in table 1. The number p of points used to build the systems of equations for kriging is 20. Although quite low when compared to the number of data points available, it is often sufficient. The number t of iterations for neural interpolation is chosen from our experience to allow convergence of the algorithm.The last column gives (an order of) the number of operations required for neural interpolation, while the one before gives this number for kriging. Remember that the problem of choosing the right p points for kriging is not taken into account to evaluate the number of operations.

Table 1. Numerical examples of the algorithmic complexity of each method.

n	m	p	t	# op. kriging	# op. neur. int.
100	1,000	20	1,000	810,000	1,000,000
10,000	1,000	20	10,000	100,800,000	10,000,000

4.2 Practical Results

The neural interpolation method has been used on synthetic and real data sets. The results are available in [10]. The synthetic data sets aimed at controlling the optimality of the results, and were therefore not too big. Some criteria were defined to assess the mean bias and optimality of the interpolations for all the neurons of a network. It was shown that the bias is nearly null if there are enough modifications of the weights, that is if the number of iterations and the gain are sufficiently high. Optimality is ensured with a very low relative error (less than 3%), and requires the same conditions as nullity of the bias. The residual relative error must be compared with the error between the experimental variance and the model used in kriging, which is generally about 8%.

Kriging has been used during the european project MEDATLAS to make an atlas of the temperature climatology field over the Mediterranean sea [2]. The aim was to produce a series of maps of the temperature climatology state of the sea, for each month and at some 29 standard depths (that is 348 maps). The number of data available was related to the depth considered, from about 260,000 in surface to less than 1,000 at the bottom. The domain to model was the whole Mediterranean basin on a grid of about 25 km step, which makes more than 15,000 grid points. To achieve this goal in a reasonable computation time, the

kriging approach was based on an adapted meshing of the data set, that allowed to select the points where it was worth computing. For the chosen points, a universal kriging method was used, based on a regional model of variance. The results were then re-gridded on a regular grid using a weight-distance linear combination of the 4 closest estimated values. An example of temperature map is given in figure 3.

A real data set was taken from the MEDATLAS project, and aimed at showing the computing time gain. The data set contained about 25,000 measures. The interpolation was required on a 204x72 points grid. The result was checked using cross validation. With this aim, the regular estimation and error grids were first reinterpolate to the data points using a classical bilinear interpolation. Then, the error between the value used to build the map and the value given by the map at each data point was computed. The mean error on all data points was found to be -0.02, which is close enough to 0 to say that the estimation is unbiased. Finally, the error on each point was compared with the predicted error. The mean value was found to be 0.82, which is close enough to 1 to say that the error map is coherent with the estimation. However, there can be cases where the estimation is very bad, although not biased, and coherent with the error computed. Therefore, it was necessary to make a graphical comparison between the maps. The map built by neural interpolation compared well with the MEDATLAS one (figure 4). The neural interpolation method required less than 1 minute of computing time, while the method used in MEDATLAS (which hopefully limited the number of points where to interpolate with the help of an adapted meshing) required more than four hours.

5 Conclusion

An original method for fast interpolation of big data sets is presented in this paper. The method relies on some basic modification of the standard Kohonen algorithm. Its unbiasness and optimality are demonstrated under hypothesis similar to those used in kriging. The method has been applied on several synthetic and actual data sets. In every cases, the results compare perfectly well with those obtained by kriging. However, the method is much faster than kriging when the data set is large, which is practically the case for actual problems. Future work will deal with taking into account an error on each data point, what kriging can do. Other studies will deal with the use of the Kohonen algorithm for data fusion and assimilation.

Acknowledgment: This work was funded by the Service Hydrographique et Océanographique de la Marine (SHOM) under contract 98.87.054. The authors wish to thank Didier Jourdan (CMO) for helpful comments and for making available a part of the MEDATLAS data set.

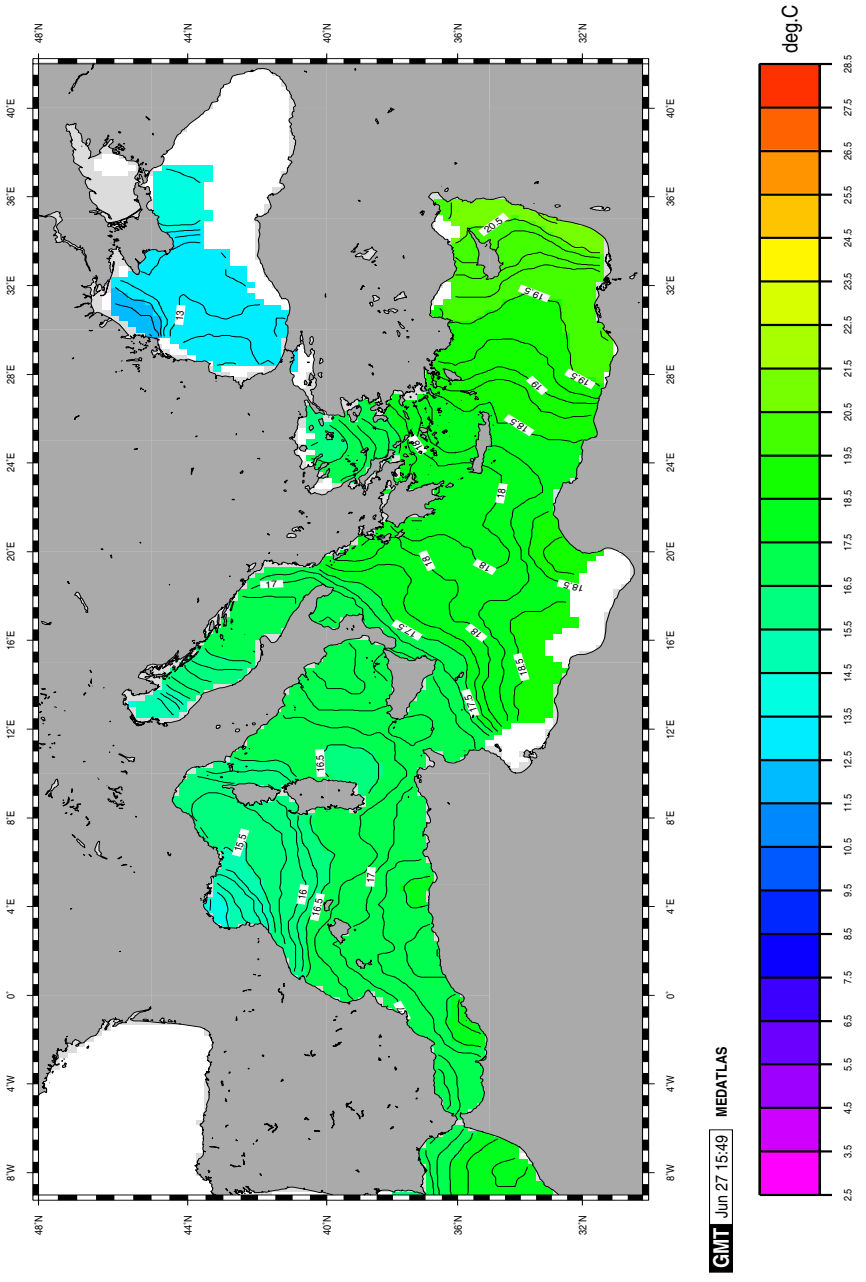


Fig. 3. MEDATLAS map of May at 10 m immersion [3].

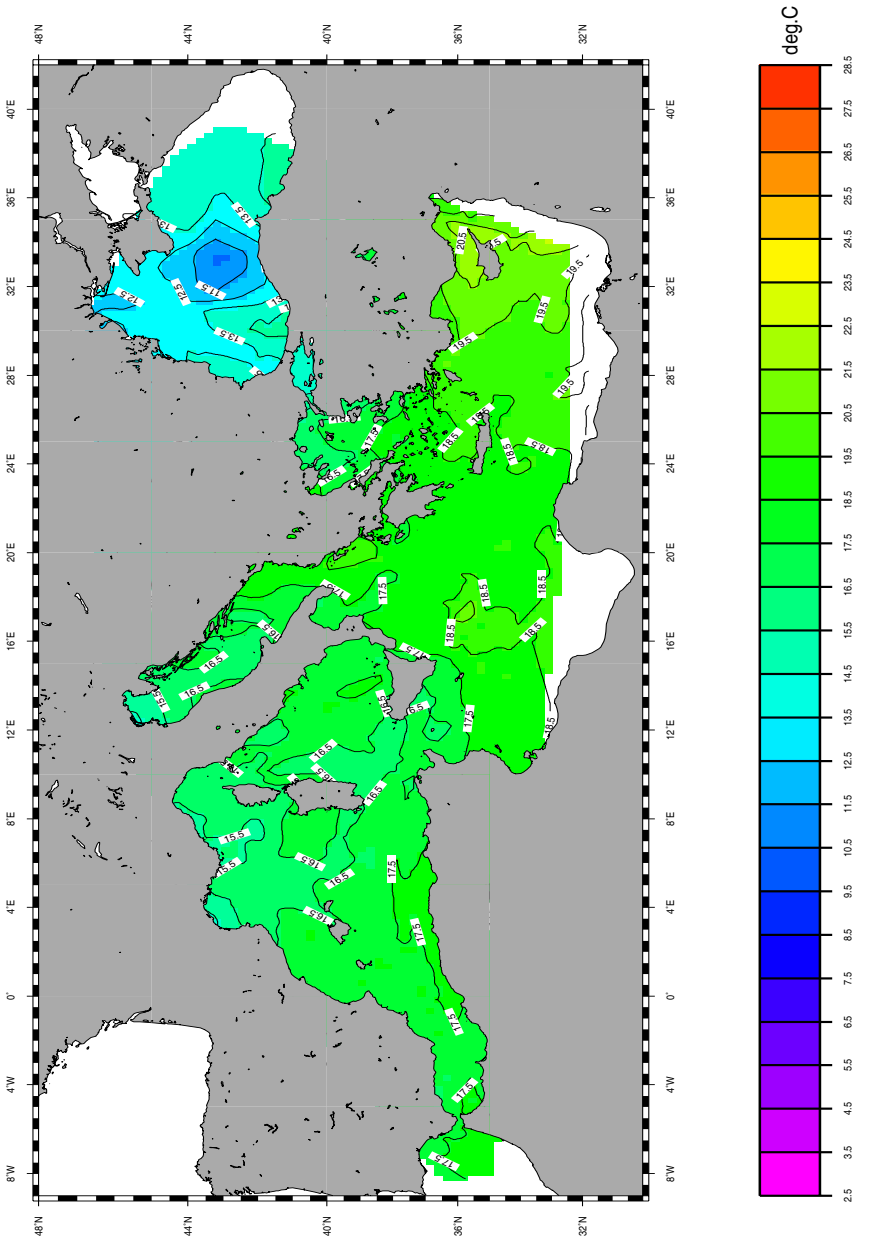


Fig. 4. Map of the Med data set produced by neural interpolation.

References

1. David, M., Crozet, D., Robb, J.M.: Automated mapping of the ocean floor using the theory of intrinsic random functions of order k. Marine Geophysical Researches **8** (1986) 49–74
2. Jourdan, D., Balopoulos, E., Garcia-Fernandez, M.-J., Maillard, C.: Objective Analysis of Temperature and Salinity Historical Data Set over the Mediterranean Basin. OCEANS'98, IEE/OES Ed. (1998) vol. 1, 82–87
3. Jourdan, D.: Bilan du projet MEDATLAS. Technical report 02P98, Service Hydrographique et Océanographique de la Marine (1998)
4. Kohonen, T.: Self-organizing maps. Springer-Verlag (1995)
5. Matheron, G.: The intrinsic random functions and their applications. Advances in Applied Probability **5** (1973) 439–468
6. Ritter, H., Schulten, K.: On the stationnary state of Kohonen's self-organizing sensory mapping. Biological Cybernetics **54** (1986) 99–106
7. Ritter, H., Schulten, K.: Convergence properties of Kohonen's topology conserving maps: Fluctuations, stability, and dimension selection. Biological Cybernetics **60** (1988) 59–71
8. Sarzeaud, O.: Les réseaux de neurones, contribution à une théorie. Ouest Editions (1994)
9. Sarzeaud, O., Stéphan, Y., Le Corre, F., Kerleguer, L.: Neural meshing of a geographical space in regard to oceanographic data location. OCEANS'94, Brest, France (1994)
10. Sarzeaud, O.: Interpolation optimale et assimilation de données par réseaux de Kohonen. Technical report OS/99003 (1999)

Steganography Using Modern Arts
(Extended Abstract)

Carlo Blundo and Clemente Galdi

Dipartimento di Informatica ed Applicazioni
Università di Salerno
84081, Baronissi (SA), Italy
{carblu,clegal}@dia.unisa.it

Abstract. In this paper we present a novel approach to information hiding. We investigate the possibility of embedding information using in some way the naturally pseudorandomness of some classes of cover-documents. In particular we provide algorithms for embedding any binary string in an image belonging to a particular class of images, the image mosaics. The algorithms presented allow different levels of security for the information hidden in the cover-document. We also show some techniques to reduce the amount of information the users have to secretly store.

1 Introduction

One of the main differences between cryptography and steganography is that each cryptographic algorithm maps a document, the plaintext, into a ciphertext that *must* be pseudo-random. Indeed it can be shown that if an encryption scheme is deterministic then it cannot be considered secure (see [3] for a more detailed description). We notice that we do not make any assumption about the document that will undergo the encryption, in the sense that we can encrypt a text as well as a binary file, that already "seems" pseudo-random.

On the other hand, each steganographic scheme hides information, the embedding-string into a document, the cover-document, without relevantly altering the information contained in it. This means that if we hide information in a non-random document, e.g. a text, the resulting document must be a a non-random document while if we hide information in a pseudo-random document, e.g. a compressed file, the result must be pseudo-random, e.g. a compressed file. In particular, the output document of the steganographic scheme must contain almost the same information of the input one.

These difference could be actually seen as the impossibility of using any cryptographic primitive in steganographic schemes.

The basic observation we make in this work is that we can use a pseudorandom-document as input of an encryption scheme and as cover-document in a steganographic scheme and both the schemes must return a pseudo-random document as output. The major problem is that steganography wants to hide to an attacker

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 140–151, 2000.
© Springer-Verlag Berlin Heidelberg 2000

the fact that an information is hidden in a document too. So, if the attacker sees that Alice sends to Bob a pseudo-random document, he will immediately detect that there are some hidden information in it. This could be avoided by properly choosing the cover-document to use. In [1] the authors show how to embed any file in a raid of pseudo-random cover-files, without relevantly altering their pseudo-randomness. On the other hand, we notice that the authors "forced", in a certain sense, the cover-documents to be pseudo-random. Indeed, no disk drive stores the information by randomizing them. This means that an attacker can immediately detect that some information are hidden in the disks.

Thus, we are searching for a class of documents that are "naturally" pseudo-random and such that their non-randomness is immediately recognizable or, in any case, checkable. This means that if we use a compressed file as cover-file, the output of the embedding function must be a compressed file too, i.e. there must exist some (public) program that correctly uncompress it. Usually, when dealing with images, the "public program" is the human visual system.

A lot of papers presents results in steganography using images as cover documents. The schemes presented in these works, embeds information into images by slightly modifying them in such a way that the human eye cannot see the difference. This is done by modifying some information in the image representation, or in its transformed representation.

Much work in the image steganography has been done on digital watermarking. The interest in these techniques is due to the growing need of copyright protection in the internet. The goal of image digital watermarking is to embed some private information into an image. This embedding procedure must satisfy two main requirement. First: the embedding procedure should not relevantly alter the original image. Second: there must exist a function that checks (or retrieves) that some information have been embedded in an image. Watermarking techniques must also guarantee the impossibility of changing the information embedded in an image without relevantly altering the image itself. This property actually states for the impossibility of changing the "ownership" information that the embedded information carries. The latest property is actually hard to asses since the images can undergo a lot of manipulation, like scaling, resizing, cropping, and so on.

One of the simplest method to hide information in an image is to alter in some way the least significant bits in a bit plane representation of the image. Examples of this techniques are [18,19]. However, these LSB methods are vulnerable in the sense that unauthorized parties could simply recover the information hidden in an image.

Due to this problem the authors in [5,15,17] developed some techniques that require the original cover-image for the retrieving phase of the information.

Another approach has been used by the authors in [14,4]. The technique described in these papers selects some pixel of the image using a pseudo random generator, and alter in some way their luminance.

In this paper we consider a particular class of pseudo-random images the Image mosaics described in the next section. We shall show that it is possible to use this class of images as cover-documents in a steganographic scheme.

The paper is organized as follows: In Section 2 we describe image databases and photomosaics. In Section 3 we give the algorithms for embedding and extracting information in an image mosaics. In Section 4 we present future works to be done.

2 Preliminaries and Notations

In this section we briefly review the terminology that we will use through this paper.

2.1 Image Databases

With the growth of world wide web, people have now access to tens of thousands of digital images, that can be collected in large databases. One of the basic problems in this case is to retrieve particular images from these databases.

According to the classic automatic information retrieval paradigm, a database is organized into documents that can be retrieved via synthetic indices, in turn organized in a data structure for rapid lookup and retrieval. The user formulates his information retrieval problem as an expression in a query language. The query is translated into the language of indices, the resulting index is matched against those in the database, and the documents containing the matched indices are retrieved.

One possible solution to the indexing problem should be to identify each image with one or more keywords. However, such an approach should avoided for some reasons. First of all, the classification should be done interactively, and this will not be, of course, a good strategy for large databases. This is not the only problem. As pointed out by the authors in [10], describing some pictorial properties of some images should be difficult, and, at the same time, some other properties should be described in different ways.

In the last years have been developed some techniques that allow the automatic indexing of image databases using "keywords" related to the represented images. Roughly, these techniques extract some information from the image and use these information as "keywords" for image indexing and lookup. A simple example of automatically extracted keyword is the hash value of the image. Of course, in these cases, the query that the user will formulate is an image itself. For more detailed description of content-based image retrieval systems see [10, 8,11]

2.2 Mosaics of Images

A classical mosaic is a set of small coloured stones arranged in such a way that if they are seen from away, they compose a larger image. This artistic expression

is based on the property of the human visual system that "sees" the colour of a region as the average colour of that region. Indeed, if the distance between two points is below a certain threshold, the human eye sees only one point whose colour is the average colour of the original ones.

The Photomosaics[1] created by R. Silvers are based on the same property (see [16] for a more detailed description). Photomosaics are mosaics in which the coloured stones are substituted by small photos. Silvers wrote a computer program that, starting from a database of small photos, called tiles images, catalogs these images according to some characteristic, like colour, shape, contrast, and many other. To create the photomosaic of a target picture, the software divides the target into macro-pixel and then substitutes each of them with an image in the database that best matches its characteristic. An example of Photomosaic[2] is presented in Figure 1.

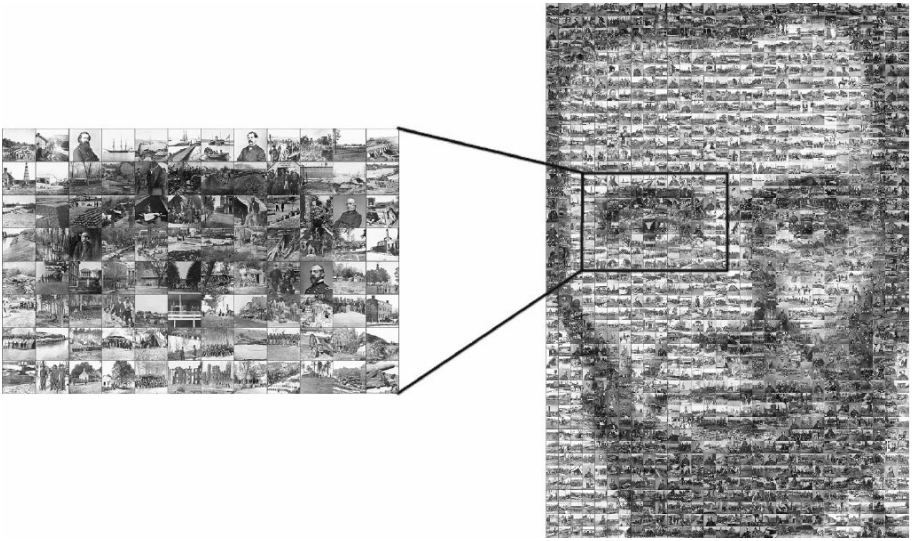


Fig. 1. Photomosaic

Other examples of image mosaics can be found in [6]. In this paper the authors describe a process to automatically generate image mosaics. The major difference between the Silvers' photomosaics and the image mosaics presented in [6] is the size of the tiles. An example of image mosaic is given in Figure 2.

Notice that currently the generation of the image mosaics is not completely automatic. This class of images is considered as an artistic expression. Thus,

[1] Photomosaic is a trademark of Runaway Technology.
[2] Image by courtesy of Runaway Technology.

Fig. 2. Image Mosaic

the authors automatically generate a first "draft" of the image mosaic and then modify it by hand in order to meet some personal criteria.

Another crucial point for automatic generation of the image mosaics is the size of the tiles. Indeed, if the size of tiles grows too much with respect the size of the image, the mosaic generated could not faithfully represent the target image (e.g., the target image resolution will decrease).

3 The Schemes

3.1 A General Scheme

In the previous section we have described what a mosaic of photos is. Here we will describe a scheme that allows to embed any secret information in such an image. For a sake of simplicity, we suppose that the mosaic $\mathcal{M}$ is a black and white picture, i.e., each tile of the mosaic can be seen as encoding a black pixel or a white one. We will later describe a generalization of this technique to the case of coloured mosaics.

More formally, we can see a mosaic $\mathcal{M}$ as an array of n tiles images, $M_1, \ldots, M_n$. Each picture M_i can be classified as belonging to one of the two classes of images $\mathcal{C}_0$ or $\mathcal{C}_1$, where $\mathcal{C}_0$ (resp., C_1) contains images that encodes a white pixel (resp., a black pixel). We suppose that Alice and Bob have agreed upon the two classes of tiles, i.e., they both hold these two databases of images and they

both hold a classification algorithm $Class$, that on input M_i, outputs an index j such that $M_i \in \mathcal{C}_j$. For the sake of simplicity, suppose that these databases have the same size $s = 2^t$, for some t (we will relax this assumption later). Moreover Alice and Bob have a standard ordering rule on each class of images, i.e., there exist four functions, known to both the players, $f_j : \{0, 1, \ldots, s-1\} \to \mathcal{C}_j$, the embedding function, and $g_j : \mathcal{C}_j \to \{0, 1, \ldots, s-1\}$, the extracting function, with $j = 0, 1$, such that for each picture P in $\mathcal{C}_j$, it results $f_j(i) = P \Longleftrightarrow g_j(P) = i$.

Let $\mathcal{X}$ be the message Alice wants to send to Bob and let $x_1, \ldots, x_\ell$ be its binary representation, where $\ell = vt$ for some integer $v \leq n$. In the following, Int, is a function that takes in input a binary string b and returns as output an integer whose binary representation is b, and Bin is its inverse function. Alice will use the algorithm presented in Figure 3.

Algorithm Embed($\mathcal{M}$,$\mathcal{X}$)

1. Partition $x_0, \ldots, x_{\ell-1}$ in v blocks, $B_0, \ldots, B_{v-1}$, of t bits each, i.e.,

$$B_i = x_{ti} x_{ti+1} \ldots, x_{ti+t-1}$$

   Notice that each block is the binary representation of an integer in $\{0, 1, \ldots, s\}$.
3. for $i = 1$ to v
   3.1 Let $j = Int(B_i)$
   3.2 Let $c = Class(M_i)$
   3.2 Substitute M_i with $f_c(j)$.
4. Output $\mathcal{M}$

Fig. 3. The Embedding Algorithm.

It can be easily seen that information that the mosaic represents, i.e., the image encoded in the mosaic, does not change, since the transformation Alice applies to the mosaic simply substitute a picture of the mosaic with another picture of the same class. It could be roughly thought as changing a bit zero in a binary string with another bit equal to zero too.

The algorithm Bob has to use to "decode" the mosaic is the presented in Figure 4.

It is immediate to see that if $v < n$, then Bob will decode the message $\mathcal{X}$ with some random elements R appended to it. To avoid this, Alice could append to $\mathcal{X}$ some special termination symbol. Moreover, notice that the scheme presented is deterministic. This means that, if $\mathcal{X} = 0^\ell$, where $\ell = vt$, then the scheme will substitute the first v tiles in the mosaic with the tiles encoding 0^t in C_0 or C_1, depending on the class the substituted tile belongs to. Of course, this situation should be avoided because, if $v < n$ the resulting mosaic will immediately reveal

Algorithm Extract($\mathcal{M}$)

1. for $i = 1$ to n
   1.1 Let $c = Class(M_i)$
   1.2 Let $e = g_c(M_i)$.
   1.3 Let $B_i = Bin(e)$
2. Output $\mathcal{X} = B_1, \ldots, B_n$.

Fig. 4. The Extracting Algorithm.

to an attacker that something strange is going on. In Sections 3.3 and 3.4 we will show how to solve this problem by randomizing the embedding scheme.

We now to evaluate the maximum length of $\mathcal{X}$ that it is possible to embed in a mosaic. We suppose that the mosaic is composed by n tiles and that the sets $\mathcal{C}_0$ and $\mathcal{C}_1$ contain s images. Since each picture "encodes" $\log s$ bits of $\mathcal{X}$, we can embed in mosaic $n \log s$ bits. We call this quantity the *capacity* of the mosaic $\mathcal{M}$.

3.2 Implementing the Function f and g

It is not hard to see that one of the basic ingredient of the algorithms are the embedding and extracting functions f and g. Recall that these function are defined as follows:

$$f_j : \{0, 1, \ldots, s-1\} \to \mathcal{C}_j \qquad g_j : \mathcal{C}_j \to \{0, 1, \ldots, s-1\}$$

A simple way to implement these functions is to "embed" them in the image database by adding to each image a field containing a (unique) number in $\{0, 1, \ldots, s-1\}$ in a preprocessing phase.

This simple process assures that to each image in a certain class is associated an unique identifier that ranges from 0 to $s-1$. Thus the f's implementation actually is a query to $\mathcal{C}$ with key equal to the input of f. On the other hand, the g's implementation is a query to $\mathcal{C}$ too. The difference between these two requests is that for the function g, the query is an image.

3.3 Hiding Private Information

In the previous sections we have shown that it is possible to embed any binary string of length ℓ in a mosaic composed by n pictures, if holding at least two classes of images with s images each and if ℓ is upper bounded by $n \log s$. The secrecy is guaranteed by the fact that an attacker should not know the database, the embedding and extracting functions. We have also shown how to simply implement these function by "embedding" them in the database itself.

This technique forces the users to locally (and privately) maintain a copy of the image databases. Notice that, there should exist some embedding and extracting functions that do not need the database modification to be implemented, (the functions should extract information from the directly from the images). If this is the case, the database could be public, but, for the privacy of the scheme presented in the previous section, the functions must still be private.

We put ourselves in a weaker condition in which the functions are implemented using some characteristic of the single image, i.e. do not depend on the database, and, at the same time, are public. To assure the privacy, we will actually embed in the mosaic an encrypted form of the message.

More precisely, an encryption scheme is a pair $(\mathcal{E}, \mathcal{D})$ of probabilistic, polynomial time algorithms where $\mathcal{E}$ is the encryption algorithm and $\mathcal{D}$ is the decryption algorithm. Recall that, for a probabilistic encryption scheme to be secure, the ciphertexts obtained by different encryption of the same plaintext must be different. Moreover, the ciphertexts obtained from the encryption must be computationally indistinguishable from a random string (see [3,7]) The encryption (resp., decryption) algorithm takes as input an encryption (resp., decryption) key k and the plaintext (resp., the ciphertext) and outputs the ciphertext (resp., the plaintext).

Let $\mathcal{X} = x_1, \ldots, x_\ell$ be the message Alice wants to send to Bob. The users will exchange the message $\mathcal{X}$ using the algorithms presented in Figure 5.

Algorithm Secure_Embed($\mathcal{M}$,$\mathcal{X}$,k)	Algorithm Secure_Extract($\mathcal{M}$,k)
1. $\mathcal{Y} = \mathcal{E}(k, \mathcal{X})$ 2. Output Embed($\mathcal{M}$,$\mathcal{Y}$)	1. $\mathcal{Y}$=Extract($\mathcal{M}$) 2. Output $\mathcal{D}(k, \mathcal{Y})$

Fig. 5. Secure Embedding and Extracting Algorithms

Even though the database and the functions are public, an attacker can extract $\mathcal{Y}$ (a pseudo-random string) using the Extract algorithm. If the encryption scheme is secure, than it is impossible for the attacker to infer information about the message $\mathcal{X}$ we have embedded in the image mosaic. Moreover, since $\mathcal{Y}$ looks completely random, the attacker cannot tell if some information has been hidden into the mosaic. Using this approach, of course, the users must share a common key k and this is the only information that the users must keep secret.

3.4 Enhancing the Security of Embedding Scheme

The protocols presented in the previous sections embeds information in an image mosaic $\mathcal{M} = M_1, M_2, \ldots, M_n$ by substituting the first v tiles in the image, for

some $v \leq n$. This means that, if an attacker believes that some information has been embedded into a mosaic, he is sure that these information are hidden in these tiles.

We have also shown how to embed private information by encrypting the message before the modification of the image. The protocol presented, in this case, does not modify the sequence of tiles changed by the embedding algorithm, even if, the amount of information that the users keep secret dramatically decreases.

The security of the scheme presented in Section 3.3, is based on the security of the underlying encryption scheme. Another simple way to improve the security of the scheme presented, is by allowing the algorithm to select the tiles to change in a pseudo-random way. As the receiver must be able to extract the information embedded in the image mosaic, the users should share a second key, k_2 ad a random seed r, that will be used in a pseudo-random number generator. This will assure the correctness of the decoding.

Algorithm Random_Embed($\mathcal{M}$,$\mathcal{X}$,r,k_1,k_2)

1. Let $\mathcal{Y} = \mathcal{E}(k_1, \mathcal{X})$.
2. Partition $\mathcal{Y}$ in v blocks, $B_1, \ldots, B_v$, of t bits each, padding zeros if necessary, i.e.,

$$B_i = y_{ti} y_{ti+1} \ldots, y_{ti+t-1}$$

3. Let Used=$\emptyset$, i=0.
4. Let $Q = q_1, q_2, \ldots, q_m$ be a sequence of m bits generated by a pseudo random number generator on input key k_2 and random seed r.
5. while $i < v$
   5.1 Let b be the first $\lceil \log n \rceil$ unused bits in Q.
   5.2 Let $j = Int(b)$
   5.3 If $j \notin$Used
      5.3.1 Substitute M_j with $f_{Class(M_j)}(B_i)$.
      5.3.2 Used=Used $\cup\{j\}$.
      5.3.3 $i \leftarrow i + 1$
6. Output $\mathcal{M}$

Fig. 6. The Embedding Algorithm with Pseudo Random Selection.

In this way, even if the attacker knows the database of the images, the embedding and the extracting function, he cannot extract any information from the image mosaic since he does not know which is the sequence of the elements he has to decode.

Notice that the algorithms presented in Figure 5 can be combined with the one presented in this section to obtain a more "powerful" scheme, presented in Figure 6, in which the user embeds the encrypted form of the message in the

image mosaic using randomly selected tiles of the original mosaic. The extracting scheme can be easily obtained by the previous one and is thus omitted.

3.5 A Note on Coloured Mosaics

In the previous sections we have discussed the embedding of private informtion in image mosaics and we have assumed that the original image is a black and white one.

However, this requirement is not necessary at all. Indeed it is possible to extend the embedding schemes presented to work with coloured mosaics.

The scheme will consider $d > 2$ classes of images $\mathcal{C}_0, \ldots, \mathcal{C}_{d-1}$ and will substitute a tile in the mosaic with another one belonging to the same class, as described in the previous sections.

4 Future Work

In the previous sections, we have analyzed some techniques allowing to embed information in image mosaics. We have based our approach on an existing class of images for which there already exists software for automatic generation.

In this section we point out the possibility to use the same techniques presented in Section 3 on a different class of cover-images. The scheme we present here does not need any database private memorization.

We now present the scheme for B/W images. Consider the cover-image as an $n \times m$-bit matrix and suppose, for a sake of simplicity, that $n = wh$ and $m = wv$, for a fixed w and for some v and h. It is thus possible to consider original cover-image as a $h \times v$ matrix of macro-pixel, each of which is a $w \times w$ matrix of pixels. A macro-pixel is said to encode black if more than $w^2/2$ pixels are black, otherwise it is said to encode white.

We can construct a "database" of macro-pixel in the following way:

- A white macro-pixel is a $w \times w$ matrix of pixels in which all but one pixels are white;
- A black macro-pixel is a $w \times w$ matrix of pixels in which all but one pixels are black.

Each macro-pixel, can be associated to a integer in the set $\{0, 1, \ldots, w^2 - 1\}$.

In Figure 7, we report a possible construction for the case of $w = 2$ with the associated integer for each macro-pixel.

Notice that not all the values of w are admissible. Indeed, as each macro-pixel encodes a value in $\{0, 1, \ldots, w^2 - 1\}$, $t = \lceil \log w^2 \rceil$ bit are necessary to represent this value. On the other hand, there exists no macro-pixel that encode the value in the set $\{w^2, w^2 + 1, \ldots, 2^t - 1\}$. This means that, if we use t bits for the embedding, there exist some binary string that cannot be injected into the mosaic. If we encode $t-1$ bit in each macro-pixel, there will be some macro-pixel that will be never used. This means that w must be equal to some power of 2.

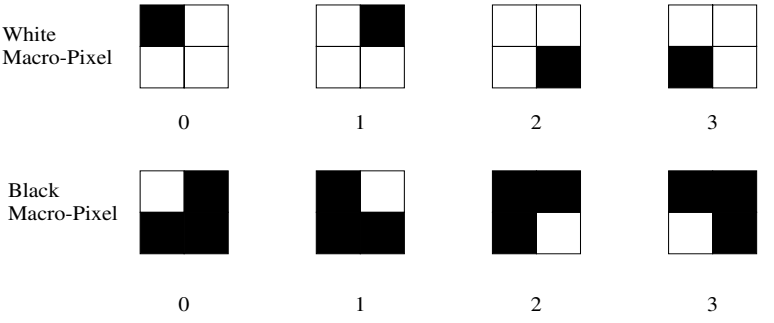


Fig. 7. Macro-Pixel Construction

Indeed, in this case, $w^2 = 2^s$ and thus we can encode in each macro-pixel exactly s bits.

It is important to remark that w's growth corresponds to a degradation of the image. We have presented a macro-pixel construction in which all but one pixels have the same colour. Of course it is possible to think to different macro-pixel constructions. For example it should be possible to require that all but two pixels have the same colour. This will, of course, raise the number of bits each macro-pixel encodes, but the image quality could decrease.

5 Conclusion

In this paper we presented a novel approach to data hiding in images. We have identified a class of images that is particularly indicated for this task. We also given some algorithms for for information hiding that allow different levels of security. The algorithms presented can be composed in order to meet the security level required.

While the algorithm presented in Section 3 are secure and do not alter the information the cover-image represents, the idea presented in Section 4 should be verified in order to give an experimental result on the image degradation.

Acknowledgements

The authors want to thank Alfredo De Santis and Paolo D'Arco for enlighting discussion that leads to this research, Robert Silvers and Runaway Technology for providing the Photomosaic in Figure 1. The authors also thanks the anonymous referee for helpful comments about the presentation of the paper.

References

1. R. Anderson, R. Needham, and A. Shamir. Steganographyc file system. In *Proceedings of 2nd Workshop on Information Hiding*, pages 73–82, 1998.
2. R.J. Anderson. Stretching the limits of steganography. In *First Workshop on Information Hiding*, pages 39–48, 1996.
3. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the des modes of operation. In *Proceedings of 38th IEEE Symposium on Foudations of Computer Science*, 1997.
4. W. Bender, D. Gruhl, N. Morimoto, and A. Lu. Techniques for data hiding. *IBM System Journal*, 3 and 4:313–336, 1996.
5. I.J. Cox, J. Kilian, T.Leighton, and T. Shamoon. Secure spread spectrum watermarking for images, audio and video. In *Proceedings of IEEE International Conference on Image Processing*, pages 243–246, 1996.
6. A. Finkelstein and M. Range. Image mosaics. In R.D. Hersch, J. André, and H. Brown, editors, *Proceedings of EP'98 and RIDT'98 Conferences*, pages 11–22, 1998.
7. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
8. C.E. Jacobs, A. Finkelstein, and D.H. Salesin. Fast multiresolution image querying. In *Proceedings of SIGGRAPH 95*, 1995.
9. A.J. Menenez, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
10. W. Niblack, R. Barber, and W. Equitz et. al. The qbic project: Querying images by content using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases, SPIE*, pages 173–187, 1993.
11. M. E. Ogle. Chabot: Retrieval from a relational database of images. *Computer*, 28(9):40–48, 1995.
12. F.A. Peticolas, R.J. Anderson, and M.G. Kuhn. Information hiding - a survey. *Proceedings of the IEEE*, 87(7):1062–1078, July 1999.
13. B. Pfitzmann. Information hiding terminology. In *Proceedings of 1st Workshop on Information Hiding*, pages 347–350, 1996.
14. I. Pitas. A method for signature casting on digital images. In *International Conference on Image Processing*, volume 3, pages 215–218, 1996.
15. C.I. Podilchuk and W. Zeng. Digital image watermarking using visual models. *Human Visual and Electronic Imaging II*, 3016:100–111, 1997.
16. R. Silvers. Photomosaics. *http://www.photomosaic.com*.
17. M.D. Swanson, B. Zhu, and A.H. Tewfik. Trasparent robust image watermarking. In *Proceedings of IEEE International Conference on Image Processing*, volume III, pages 211–214, 1996.
18. R. van Schyndel, A. Tirkel, and C. Osborne. A digital watermarking. In *Proceedings of IEEE International Conference on Image Processing*, pages 86–90, 1994.
19. R.B. Wolfgang and E.J. Delp. A watermarking for digital images. In *Proceedings of IEEE International Conference on Image Processing*, pages 219–222, 1996.

Trade-Offs between Density and Robustness in Random Interconnection Graphs*

Philippe Flajolet[1], Kostas Hatzis[2,3], Sotiris Nikoletseas[2,3], and Paul Spirakis[2,3]

[1] Algorithms Project, INRIA Rocquencourt, France
Philippe.Flajolet@inria.fr
[2] Computer Technology Institute (CTI), Patras, Greece
{hatzis, nikole, spirakis}@cti.gr
[3] Computer Engineering and Informatics Department,
Patras University, Greece.

Abstract. Graphs are models of communication networks. This paper applies combinatorial and symbolic-analytic techniques in order to characterize the interplay between two parameters of a random graph: its density (the number of edges in the graph) and its robustness to link failures, where robustness here means multiple connectivity by short disjoint paths. A triple (G, s, t), where G is a graph and s, t are designated vertices, is called $\ell - robust$ if s and t are connected via at least two edge-disjoint paths of length at most ℓ. We determine here the expected number of ways to get from s to t via two edge-disjoint paths in the random graph model $G_{n,p}$. We then derive bounds on related threshold probabilities $p_{n,\ell}$ as functions of ℓ and n.

1 Introduction

In recent years the development and use of communication networks has increased dramatically. In such networks, basic physical architecture combined with traffic congestion or operating system decisions, result in a certain, dynamically changing geometry of the graph of interconnections. We adopt the random graph model of $G_{n,p}$ (see [3], [4]) to capture link availability with probability p (independently for each link) in a fully connected graph. Even in such a simple network model, it is interesting to investigate the trade-off between its *density* (the number of edges) and its *robustness* to link failures. The existence of alternative paths in such graphs models desired reliability and efficiency properties, such as the ability to use alternative routes to guide packet flow in ATM networks or even improve the efficiency of searching robots on the World Wide Web, in the sense of an increased multiconnectivity of its hyperlink structure.

Given a triple (G, s, t), where G is a $G_{n,p}$ random graph and s, t are two of its nodes, a natural notion of robustness is to require at least two edge disjoint

* This work was partially supported by the EU Projects ESPRIT LTR ALCOM-IT, IST-FET-OPEN ALCOM-FT, IMPROVING RTN ARACNE and the Greek GSRT Project PENED-ALKAD.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 152–168, 2000.
© Springer-Verlag Berlin Heidelberg 2000

paths of short length (say, exactly ℓ or at most ℓ) between s and t, so that connectivity by short paths survives, even in the event of a link failure. We next give the following formal definition of $\ell - robustness$.

Definition 1 (ℓ-robustness). *A random graph G of the model $G_{n,p}$ is $\ell -$ robust for two vertices s, t when there are two edge-disjoint paths of length at most ℓ between s, t in G.*

In this work, we investigate the expected number of such paths between two vertices of the random graph, as well as bounds for the threshold probability $p_{n,\ell}$ (as functions of ℓ and n) for the existence of such paths in the $G_{n,p}$ random graph G.

Although $G_{n,p}$ has been extensively studied (see [1], [4], [14]), some questions of existence of multiple paths, which are vertex or edge disjoint between specific vertices have not been investigated till recently. The theory of random graphs began with the celebrated work of Erdös and Renyi ([7]) in 1959 and by now researchers know lot about the probable structure of these objects (see, e.g., the birth of the giant connected component in [11]). In this context we remark that the question of existence of many *vertex disjoint* paths of small length has been investigated by Nikoletseas et al in [13].

With respect to the corresponding question of the existence of many *edge* disjoint paths, we refer to the fundamental work of Broder et al ([5]). In that work, the authors show the existence of many edge disjoint paths in dense (with high probability connected) random graphs. However, the above work does not address the question of the existence of many edge disjoint paths *of a certain length*. Also, the estimation of the *precise number* of such paths (as a function of the density of the graph) still remains open.

It turns out that even the enumeration of paths among the vertices 1 and n that avoid all edges of the graph $(1, 2 \ldots, n)$ but pass through its vertices, is a non-trivial task. In fact, such an enumeration is a special case of enumerating permutations $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ of $(1, 2, \ldots, n)$ where certain gaps $\sigma_{i+1} - \sigma_i$ are forbidden. In our case, $\sigma_{i+1} - \sigma_i$ must not be in the set $\{-1, 1\}$.

In this work, we provide a precise estimate of the expected number of unordered pairs of paths in a random graph that connect a common source to a common destination, and have no edge in common, though they may share some nodes. Thus, for any given set of values of n, p, ℓ (where ℓ represents the length) we estimate precisely the mean number of avoiding pairs in graphs of a given size. This leads to a tight bound for the probability of non-existence of such paths between any fixed pair of nodes and also to a bound for the probability of the existence of many pairs of edge disjoint paths of length ℓ.

In order to achieve this, we devise a finite state mechanism that describes classes of permutations with free places and exceptions. The finite-state description allows for a direct construction of a multivariate generating function. The generating function is then subjected to an integral transform that implements an inclusion-exclusion argument and an explicit enumeration result; see Theorems 1 and 2. This enables us to quantify the trade-off between ℓ-robustness

(as defined above) and the density of the graph (i.e. the number of its edges). The originality of our approach consists in introducing in this range of problems methods of analytic combinatorics and recent research in automatic analysis (based on symbolic computation). For context, see [8], [9], [6]. Additional threshold estimates regarding properties of multiple source-destination pairs are discussed in the last two sections of the paper.

Summary of results: From earlier known results ([4], [13]) and this paper, a picture of robustness under the $G_{n,p}$ model emerges. (As usually in random graph theory, various regimes for $p = p(n)$ are considered). Start with an initially totally disconnected graph, corresponding to $p = 0$. As p increases, the graph becomes connected near the connectivity threshold $p_c(n) \simeq (\log n)/n$. All the following results hold for any integer ℓ: $6 \leq \ell < n$. Any fixed s, t pair (or equivalently a random s, t pair, given the invariance properties of $G_{n,p}$) *is "likely" to be ℓ-robust* when p crosses the value

$$p_{n,\ell}(n) = 2^{\frac{1}{2\ell}}\ n^{-1+\frac{1}{\ell}}$$

(see Theorem 3 and Equation 6). (Here "likely" means that the expected number of edge-disjoint pairs is at least 1 when $n \to \infty$). As long as $p \leq p_L(n, \ell)$, where

$$p_L(n, \ell) = n^{-1+\frac{1}{\ell}} \left(\log \frac{n^2}{\log n} \right)^{\frac{1}{\ell}}$$

we know, w.h.p., the *existence* of s, t pairs that are *not* connected by short (of length at most ℓ) paths; see Theorem 4. (The function $p_L(n)$ is in fact a threshold for diameter). However, one only needs a tiny bit more edges, namely $p \geq p_U$:

$$p_U(n, \ell) = n^{-1+\frac{1}{\ell}}\ 2 \left(\log \left(n^2 \log n \right) \right)^{\frac{1}{\ell}}$$

to ensure that *almost all* s, t-pairs are ℓ-robust; see Theorem 5.

2 The Enumeration of "Avoiding" Configurations

The problem at hand is that of estimating the expected number of "avoiding pairs" of length ℓ between a random source and a random destination in a $G_{n,p}$ random graph G. (An avoiding pair of length ℓ means an unordered pair of paths, each of length ℓ, that connect a common source to a common destination, and have no edge in common though they may share some nodes). This problem first necessitates the solution of enumeration problems that involve two major steps:

— Enumerate simple paths called "avoiding permutations" of length ℓ that can be viewed as hamiltonian paths on the set of nodes $[1, \ldots, \ell + 1]$, connecting the source 1 and the destination $\ell + 1$, and having no edge of type $[i, i + 1]$ or $[i, i - 1]$.

— Enumerate so-called "avoiding paths", that are simple paths allowed to contain outer nodes taken from outside the segment $[1, \ldots, \ell+1]$. This situation is closer to the random graph problem since it allows nodes taken from the pool of vertices available in the graph $G \in G_{n,p}$.

The first problem is of independent combinatorial interest as it is equivalent to counting special permutations with restrictions on adjacent values. It also serves as a way to introduce the methods needed for the complete random graph problem. Both problems rely on the inclusion-exclusion principle that is familiar from combinatorial analysis and counting by generating functions (GF's). Applications of these results to the $G_{n,p}$ model are treated in the next section.

2.1 Symbolic Enumeration Methods

We use here a symbolic approach to combinatorial enumeration, according to which many general set-theoretic constructions have direct translations over generating functions. A specification language for elementary combinatorial objects is defined for this purpose. The problem of enumerating a class of combinatorial structures then simply reduces to finding a proper specification, a sort of a formal grammar, for the class in terms of basic constructions. (This general method has been carefully explained in the fundamental work of F. Chyzak, Ph. Flajolet and B. Salvy ([5])).

In this framework, classes of combinatorial structures are defined either iteratively or recursively in terms of simpler classes by means of a collection of elementary combinatorial constructions. The approach followed resembles the description of formal languages by means of context-free grammars, as well as the construction of structured data types in classical programming languages.

The approach developed here is direct, more "symbolic", as it relies on a specification language for combinatorial structures. It is based on so-called admissible constructions that have the important feature of admitting direct translations into generating functions. We specifically examine constructions whose natural translation is in terms of ordinary generating functions.

The ordinary generating function (OGF) of a sequence $\{A_n\}$ is, we recall, $A(z) = \sum_{n=0}^{\infty} A_n \cdot z^n$.

Definition 2 (Admissible Constructions). *Assume that Φ is a binary construction that associates to two classes of combinatorial structures B and C a new class $A = \Phi(B, C)$ in a finite way (each A_n depends on finitely many of the B_n and C_n). The Φ is admissible iff the counting sequence $\{A_n\}$ of A is a function of the counting sequences $\{B_n\}$ and $\{C_n\}$ of B and C only: $\{A_n\} = \Xi[\{B_n\}, \{C_n\}]$. In that case, there exists a well defined operator Ψ relating the corresponding ordinary generating functions: $A(z) = \Psi[B(z), C(z)]$.*

In this work, we will basically use three important constructions: union, product and sequence, which we describe below.

Definition 3 (Union Construction). *The disjoint union A of two classes B, C is the union (in the standard set-theoretic sense) of two disjoint copies, B^o and C^o, of B and C. Formally, we introduce two distinct "markers" ϵ_1 and ϵ_2, each of size zero, and define the (disjoint) union $A = B + C$ of B, C by $B + C = (\{\epsilon_1\} \times B) \cup (\{\epsilon_2\} \times C)$. The ordinary generating function is clearly $A(z) = B(z) + C(z)$. We represent the disjoint union construction by* Union.

Definition 4 (Product Construction). *If construction A is the cartesian product of classes B and C ($A = B \times C$), then, considering all possibilities, the counting sequences corresponding to A, B, C are related by the convolution relation: $A_n = \sum_{k=0}^{n} B_k \cdot C_{n-k}$ and the ordinary generating function is clearly $A(z) = B(z) \cdot C(z)$. We represent the product construction by* Prod.

Definition 5 (Sequence Construction). *If C is a class of combinatorial structures then the sequence class $\mathcal{G}\{\mathcal{C}\}$ is defined as the infinite sum $\mathcal{G}\{\mathcal{C}\} = \{\epsilon\} + \mathcal{C} + (\mathcal{C} \times \mathcal{C}) + \cdots$ with ϵ being a "null structure", meaning a structure of size 0. The null structure plays a role similar to that of the empty word in formal language theory and the sequence construction is analogous to the Kleene star operation (C^*). The ordinary generating function is clearly given by $A(z) = 1 + B(z) + B^2(z) + \cdots = \frac{1}{1-B(z)}$ where the geometric sum converges in the sense of formal power series since $[z^0]B(z) = 0$. We represent the sequence construction by* Sequence.

2.2 Avoiding Permutations

An avoiding permutation of length ℓ is a sequence $\tau = [\tau_1, \tau_2, \ldots, \tau_\ell, \tau_{\ell+1}]$ that is a permutation of $[1, \ldots, \ell+1]$ and that satisfies the following conditions: $\tau_1 = 1$, $\tau_{\ell+1} = \ell + 1$, and $\tau_{i+1} - \tau_i \neq \pm 1$ for all i such that $1 \leq i \leq \ell$. Clearly, such a permutation encodes a path from 1 to $\ell + 1$ that has no edge in common with the graph $(1, 2, \ldots, \ell + 1)$. The parameter $\ell + 1$ is referred to as the size. There is no avoiding permutation for sizes $2, 3, 4, 5$. Surprisingly, the first nontrivial configurations occur at size 6, where the 2 possibilities are $[1, 4, 2, 5, 3, 6]$ and $[1, 3, 5, 2, 4, 6]$, while for size 7, there are 10 possibilities

$[1, 3, 6, 4, 2, 5, 7], [1, 3, 5, 2, 6, 4, 7], [1, 4, 6, 2, 5, 3, 7], [1, 4, 2, 6, 3, 5, 7], [1, 5, 3, 6, 4, 2, 7],$
$[1, 5, 3, 6, 2, 4, 7], [1, 5, 2, 4, 6, 3, 7], [1, 4, 6, 3, 5, 2, 7], [1, 6, 3, 5, 2, 4, 7], [1, 6, 4, 2, 5, 3, 7].$

The goal in this subsection is to determine the number Q_n of avoiding permutations of size n (that is, of length $n - 1$). We prove:

Theorem 1. *Avoiding permutations have ordinary generating function*

$$Q(z) := \sum_n Q_n z^n = \frac{\left(z - 1 + e^{\frac{1+z}{z(z-1)}}\, Ei\left(1, \frac{1+z}{z(z-1)}\right)\right) z}{(z-1)(1+z)} =$$

$$\frac{z\left(z + z\, hypergeom\left([1,1], [\,], -\frac{z(z-1)}{1+z}\right) + 1\right)}{(1+z)^2}$$

where Ei *is the exponential integral and* hypergeom *represents the hypergeometric series. Equivalently,* Q_n *is expressible as a double binomial sum:*

$$Q_{n+2} = (-1)^{n-1} + \sum_{k_2=0}^{n} \sum_{k_1=0}^{n-k_2} (-1)^{k_1+k_2} (n-k_1-k_2)! \binom{n-k_1-k_2}{k_1} \binom{n+1-k_1}{k_2}$$

Proof. By the inclusion-exclusion principle, we need to determine the number $F_{n,j}$ of permutations $[\tau_1 = 1, \tau_2, \ldots, \tau_{n-1}, \tau_n = n]$, with at least j "exceptions", among which j distinguished, that are successions of values of the form $\tau_j - \tau_{j-1} = \pm 1$. The number of permutations with no exception is then:

$$Q_n = \sum_{j=0}^{n-1} (-1)^j F_{n,j} \tag{1}$$

A permutation with exceptions can be regarded as including a subcollection of "exceptional" edges that belong to the graph with edges $(1,2), (2,3), \ldots, (n-1, n)$. If we scan from left to right and group such exceptions by blocks, we get a *template*; a template thus represents a possible pattern of exceptional edges.

A template can be defined directly as made of blocks that are either: (i) isolated points; (ii) contiguous unit intervals oriented left to right (LR); (iii) contiguous unit intervals oriented right to left (RL). There is the additional constraint that the first and last blocks cannot be of type RL. For instance, for $n = 13$, the template $[\,[1,2,3],\ [4],\ [5,6],\ [7],\ [8],\ [11,10,9],\ [12,13]\,]$ will correspond to any permutation that has successions of values (in the cycle traversal) $1,2\,;\ 2,3\,;\ 5,6\,;\ 11,10\,;\ 10,9\,;\ 12,13$ as distinguished exceptions to the basic constraint of avoiding permutations.

We next provide the combinatorial specification for avoiding permutations.

Let $\{a, b\}$ be a binary alphabet. We now describe the grammar of templates.

The collection of strings beginning and ending with a letter a is described by the following rule:

$$sp0 := S = \text{Prod}(\text{Sequence}(\text{Prod}(a,\ \text{Sequence}(b))),\ a)$$

(It suffices to decompose according to each occurrence of the letter a). Now, the thre types of blocks in a template are described by the following rules:

$$sp1 := \text{Prod}(\textit{begin_blockP},\ Z,\ \textit{end_blockP})$$
$$sp2 := \text{Prod}(\textit{begin_blockLR},\ Z,\ \text{Sequence}(\text{Prod}(\textit{mu_length},\ Z),\ 1 \leq \textit{card}),\ \textit{end_blockLR})$$
$$sp3 := \text{Prod}(\textit{begin_blockRL},\ \text{Sequence}(\text{Prod}(\textit{mu_length},\ Z),\ 1 \leq \textit{card}),\ Z,\ \textit{end_blockRL})$$

Clearly, $sp2$ and $sp3$ are combinatorially isomorphic. For reasons related to the application of the inclusion-exclusion argument, we keep track of the size (number of nodes $l = 1$ of the basic interval graph, denoted by *card*) as well as of the length of blocks and of their LR or RL character, denoted by *mu_length*. Then, the grammar of templates is completed by substituting into $sp0$

$$a = \text{Union}(sp1, sp2) \quad \text{and} \quad b = sp3$$

Let $F_{n,k,j}$ be the number of templates with size n, k blocks and j exceptional edges. Then, counting the number of ways of linking blocks together, yields:

$$F_{n,j} = \sum_k F_{n,k,j}\, \phi(k) \tag{2}$$

where $\phi(k)$ is the Gamma integral:

$$\phi(k) \equiv (k-2)! = \int_0^\infty e^{-u}\, u^k\, \frac{du}{u^2}$$

for $k \geq 2$, and $\phi(1) = 1$ (since any such linking is determined by an arbitrary permutation of the $k-2$ intermediate blocks). Observe that the extension of ϕ by linearity to an arbitrary series $\mathrm{h}(u)$ in u is given by

$$\phi(\mathrm{h}(u)) = \int_0^\infty \frac{e^{-u}\left(\mathrm{h}(u) - (u-u^2)\left(\frac{\partial}{\partial u}\mathrm{h}(u)\right)_{u=0}\right)}{u^2}\, du$$

That is to say, we just replace in expansions $u \to u^2$ and apply the Euler integral

$$\int_0^\infty e^{-u} u^k du = k!$$

Thus, with $\mathrm{F}(z,u,v) = \sum F_{n,k,l}\, z^n u^k v^l$,the OGF $\mathrm{Q}(z) = \sum Q_n z^n$ satisfies

$$\mathrm{Q}(z) = \phi(\mathrm{F}(z,u,-1))$$

Thus, from (1) and (2) above, everything boils down to obtaining the $F_{n,k,j}$.

Template enumeration. The approach to determining the sequence $F_{n,k,j}$ consists in introducing the trivariate GF, which immediately results from the above combinatorial specification.

$$F(z,u,v) = \sum_{n,k,\ell} F_{n,k,\ell}\; z^n u^k v^l$$

There, z records size, u records the total number of blocks (needed for subsequent permutation enumerations since blocks should be chained to each other), and v records the total length of LR or RL blocks (the number of distinguished exceptions needed for inclusion-exclusion).

We now carefully employ the generating functions for the union, product and sequence constructions in the grammar rules of the combinatorial specification for avoiding permutations defined above.

The set of words made of a's and b's that start and end with an a is described symbolically by

$$W = \frac{1}{1 - \frac{a}{1-b}} \cdot a \tag{3}$$

This is because $(1-f)^{-1} = 1 + f + f^2 + f^3 + \cdots$ generates symbolically all sequences of objects of type f. Thus, W represents a sequence of objects of type

$\frac{a}{1-b}$ that start with an a. On the other hand, $\frac{a}{1-b}$ represents a sequence of objects of type b that end with an a. Take now the three types of blocks: isolated, LR, and RL. The GF's are, respectively, z, $LR(z) = z^2/(1-z)$, $RL(z) = z^2/(1-z)$. This is because isolated points are always of size 1, while LR and RL objects must be of size at least 2 (we have thus to multiply with z^2). Since the first and the last blocks can only be isolated points or LR blocks, the univariate GF for blocks is obtained by substituting a by $z + LR$ (isolated point or LR block) and b by RL in W. Thus we get the following trivariate GF:

$$F(z,u,v) = \left(1 - \frac{u\left(z + \frac{z^2 v}{1-vz}\right)}{1 - \frac{uz^2 v}{1-vz}}\right)^{-1} \cdot u\left(z + \frac{vz^2}{1-vz}\right) =$$

$$= -\frac{uz(-1 + vz + uz^2 v)}{1 - 2vz - uz + v^2 z^2 + v^2 z^3 u}$$

Path counting. For the inclusion-exclusion argument, it is easy to observe that the desired sum $\sum_{n,k,\ell} F_{n,k,\ell}\, z^n u^k (-1)^\ell$ corresponds to the specialization $F(z,u,-1)$. This yields:

$$F(z,u,-1) = -\frac{uz(-1 - z - uz^2)}{1 + 2z - uz + z^2 + z^3 u} \tag{4}$$

Application of the ϕ-transformation (that counts the number of ways to connect the blocks) requires the modified form and so we get:

$$F(z,u,-1) = \frac{zu^2(uz^2 + 2z - uz + 1)}{(1+z)(uz^2 + z - uz + 1)}$$

The corresponding ordinary generating function is

$$Q(z) = \int_0^\infty \frac{z(uz^2 + 2z - uz + 1)}{(1+z)(uz^2 + z - uz + 1)} \cdot e^{-u}\, du$$

The quantity $Q(z)$ can be expressed in terms of the exponential integral

$$\int_0^\infty e^{-u}\, u^k\, du = k!$$

in the following closed form

$$Q(z) := \frac{\left(z - 1 + e^{\frac{1+z}{z(z-1)}}\, \mathrm{Ei}\left(1, \frac{1+z}{z(z-1)}\right)\right) z}{(z-1)(1+z)}$$

Since one deals with ordinary generating functions, this is to be taken as a formal (asymptotic) series. Note also that the exponential integral (Ei) involves the divergent series of factorials $\sum_{n=0}^\infty n!(-y)^{(-n-1)}$ which is also a hypergeometric series. This gives rise to a general conversion procedure from exponential

integrals to hypergeometric forms. Hence, another closed form for the OGF of the Q_n is

$$Q(z) := \frac{z\left(z + z\,\text{hypergeom}\left([1,1],[\,],-\frac{z(z-1)}{1+z}\right)+1\right)}{(1+z)^2}$$

Thus, with $\mathrm{F}(z,u,v) = \sum F_{n,k,l}\, z^n u^k v^l$ and for OGF $\mathrm{Q}(z) = \sum Q_n z^n$, by recalling that $\mathrm{Q}(z) = \phi(\mathrm{F}(z,u,-1))$ we get the expression for $Q(z)$ as stated in the Theorem. The expression can then be expanded using the binomial theorem, and double combinatorial sums result for coefficients. □

Though they have no direct bearing on the graph problem at hand, we mention two interesting consequences of this theorem.

Corollary 1. *The quantities Q_n satisfy the recurrence*

$$(n+1)Q_n + Q_{n+1} - 2nQ_{n+2} + 4Q_{n+3} + (n+3)Q_{n+4} - Q_{n+5} = 0,$$

where $Q(0) = 0$, $Q(1) = 1$, $Q(2) = Q(3) = Q(4) = Q(5) = 0$ and the asymptotic estimate

$$\frac{Q_n}{(n-2)!} = e^{-2}\left(1 + O\left(\frac{1}{n}\right)\right)$$

Proof. To get the recurrence relation, we use the following holonomic descriptions (introduced by Zeilberger), that is sequences that satisfy linear recurrences with polynomial coefficients:

$$\left(z^4+z^5+4z^3-1-z+4z^2\right)\mathrm{Y}(z) + \left(-2z^4+z^2+z^6\right)\left(\frac{\partial}{\partial z}\mathrm{Y}(z)\right) -$$

$$-z^4 - 2z^3_C_0 - 2z^3 - z^4_C_0 + _C_0 z^2 - z^5 + z - z^2 = 0$$

where $Y(z) = Q(z)$. By putting $C_0 = 1$, we get:

$$(z^4+z^5+4z^3-1-z+4z^2)\mathrm{Y}(z)+(-2z^4+z^2+z^6)\left(\frac{\partial}{\partial z}\mathrm{Y}(z)\right)-2z^4-4z^3-z^5+z=0$$

We can now get (by elementary properties of the z-transform) the following transformation to a linear recurrence:

$$u(0) = 0,\ u(1) = 1,\ u(2) = 0,\ u(3) = 0,\ u(4) = 0,\ u(5) = 0,$$

$$(n+1)u(n) + u(n+1) - 2n\,u(n+2) + 4u(n+3) + (n+3)u(n+4) - u(n+5) = 0$$

We note that this provides an algorithm that uses a linear number of arithmetic operations to determine the quantities Q_n. By using the following principle based on the generating function method:

$$coeff_{z^n}\,\text{hypergeom}\left([1,1],[\,],z+dz^2+\mathrm{O}\left(z^3\right)\right) = n!\,e^d(1+\mathrm{o}(1))$$

provided that the argument of the hypergeometric is a function that is analytic at the origin, we have proved

$$Q(z) = \frac{z\left(z + z\,\text{hypergeom}\left([1,1],[\,],-\frac{z(z-1)}{1+z}\right)+1\right)}{(1+z)^2}$$

and since

$$-\frac{z(z-1)}{1+z} = z - 2z^2 + 2z^3 - 2z^4 + 2z^5 + \mathrm{O}(z^6)$$

we have proved that the asymptotic proportion of legal permutations is *exactly* equal to e^{-2}. □

The recurrence above implies the non-obvious fact that the number of avoiding permutations Q_n are computable in linear time. The asymptotic estimate extends properties known for permutations with excluded patterns (e.g., derangements have asymptotic density e^{-1}). Consequently, a nonzero proportion (about 13.53%) of all permutations that start with 1 and end with n are avoiding.

2.3 Avoiding Paths

We consider now the problem of counting the number $Q_{n,j}$ of avoiding paths of type (n, j), where n is the size (the number of nodes) and j is the number of "outer nodes". Such avoiding paths are defined by the fact that they satisfy the basic constraints of avoiding permutations regarding the base line $(1, 2, \ldots, n)$, but contain in addition j outer nodes taken to be indistinguishable (unlabelled) and conventionally represented by the symbol '$\star$'. For instance, for types $(n, j) = (3, 1), (4, 1), (4, 2)$, the listings are respectively

$$\{[1, \star, 3]\} \qquad \{[1, 3, \star, 4], [1, \star, 2, 4]\} \qquad \{[1, \star, \star, 4]\}$$

Theorem 2. *The number of avoiding paths is expressible as*

$$Q_{n+2,j} = \sum_{k_2=0}^{n-j} \sum_{k_1=0}^{n-j-k_2} (-1)^{k_1+k_2} (n-k_1-k_2)! \binom{n-j-k_1-k_2}{k_1} \cdot \cdot \binom{n-j+1-k_1}{k_2} \binom{n-k_1-k_2}{j}^2$$

(where $j \geq 0$)

Proof. We first define templates on which an inclusion-exclusion argument is applied. The specifications are a simple modification of the templates associated to avoiding permutations.

Let $\{a, b, x\}$ be a ternary alphabet. We now define the grammar of templates.

The collection of strings beginning with a and containing only one x that occurs at the end is described by the rule:

$$sp0 := S = \text{Prod}(\text{Sequence}(\text{Prod}(a, \text{Sequence}(b))), x)$$

(It suffices to decompose according to each occurrence of the letter a). We first need so-called "outer points" that are taken from outer space:

$$Outerpoints := \text{Sequence}(\text{Prod}(Z, mu_outerpoint))$$

We also need "inner points":

$$Innerpoints := \text{Sequence}(\text{Prod}(Z, mu_innerpoint))$$

Size is defined as the cumulative number of points in the pair of paths that underlies an avoiding path in the sense above: it is thus equal to the length of the avoiding path plus the number of $\star$ symbols corresponding to the outer nodes. We thus introduce a special notation for nodes of the integer line that are shared by the two paths:

$$Z2 := \text{Prod}(Z, Z)$$

Now, the three types of blocks are described by the following rules:

$$sp1 := \text{Prod}(mu_block,\ Z2,\ Outerpoints,\ Innerpoints)$$
$$sp2 := \text{Prod}(mu_block,\ Z2,\ \text{Sequence}(\text{Prod}(mu_length,\ Z2), \text{card} \geq 1),\ Outerpoints,\ Innerpoints)$$
$$sp3 := \text{Prod}(\text{Sequence}(\text{Prod}(mu_length,\ Z2), \text{card} \geq 1),\ Z2,\ Outerpoints,\ Innerpoints,\ mu_block)$$

(Clearly, $sp2$ and $sp3$ are combinatorially isomorphic). The blocks that can occur at the end are of type x and can only be of type $sp1$ or $sp2$ but without outer points nor inner points.

$$sp1x := \text{Prod}(mu_block,\ Z2)$$
$$sp2x := \text{Prod}(mu_block,\ Z2,\ \text{Sequence}(\text{Prod}(mu_length,\ Z2), 1 \leq card))$$

The above grammar is completed (to give S) by substituting into $sp0$

$$a = \text{Union}(sp1, sp2)$$
$$b = sp3 \quad \text{and}$$
$$x = \text{Union}(sp1x, sp2x)$$

The 5-variate GF immediately results from the above specification:

$$\text{F}(z, u, v, w_1, w_2) := \frac{-u\,z^2(-1 + z\,w_2 + z\,w_1 - z^2\,w_1\,w_2 + v\,z^2 - v\,z^3\,w_2 - v\,z^3\,w_1 + v\,z^4\,w_1\,w_2 + u\,z^4\,v)}{1 - z(w_2 + w_1) + z^2(w_1\,w_2 - u) + v\,z^2(-2 + 2z\,w_1 - 2\,z^2\,w_1\,w_2 + v\,z^2(1 - z\,w_2 - z\,w_1 + z^2\,w_1\,w_2 + z^2\,u))}$$

where u, v, w_1, w_2 represent the blocks, the length, the outer nodes and the inner nodes, respectively.

For inclusion-exclusion, we set $v = -1$. Application of the ϕ-transformation (that counts the number of ways to connect the blocks) requires the modified form

$$F(z,u,-1,w_1,w_2) := \frac{u^2\,z^2(1+2z^2-z^3\,w_1+u\,z^4+z^4\,w_1\,w_2-z^3\,w_2-u\,z^2-z\,w_2-z\,w_1+z^2\,w_1\,w_2)}{(1+z^2)(z^4\,w_1\,w_2+u\,z^4-z^3\,w_2-z^3\,w_1+z^2\,w_1\,w_2+z^2-u\,z^2-z\,w_2-z\,w_1+1)}$$

The ordinary generating function is here

$$Q(z) := \int_0^\infty \frac{z^2(1+2z^2-z^3\,w_1+u\,z^4+z^4\,w_1\,w_2-z^3\,w_2-u\,z^2-z\,w_2-z\,w_1+z^2\,w_1\,w_2)\,e^{-u}}{(1+z^2)(z^4\,w_1\,w_2+u\,z^4-z^3\,w_2-z^3\,w_1+z^2\,w_1\,w_2+z^2-u\,z^2-z\,w_2-z\,w_1+1)}\,du$$

And this can be expressed in terms of the exponential integral as follows:

$$Q(z) := z^2\cdot e^{-\frac{(1+z^2)(w_2+w_1)}{z(z-1)(1+z)}}\cdot \frac{\mathrm{Ei}\left(1,\frac{z^4\,w_1\,w_2-z^3\,w_2-z^3\,w_1+z^2\,w_1\,w_2+z^2-z\,w_2-z\,w_1+1}{z^2(z^2-1)}\right)e^{\frac{(1+z^2)(z^2\,w_1\,w_2+1)}{z^2(1-z^2)}}}{(z^2-1)(1+z^2)}+$$
$$+z^2\cdot e^{-\frac{(1+z^2)(w_2+w_1)}{z(z-1)(1+z)}}\cdot\frac{z^2e^{\frac{(1+z^2)(w_2+w_1)}{z(1-z^2)}}-e^{\frac{(1+z^2)(w_2+w_1)}{z(1-z^2)}}}{(z^2-1)(1+z^2)}$$

Again, there is an "explicit form" of the OGF of the problem

$$Q(z) := z^2\cdot\frac{z^3(z\,w_1\,w_2-w_2-w_1)+z^2\,\mathrm{hypergeom}\left([1,1],[\,],-\frac{z^2\,(z-1)(1+z)}{(1+z^2)(z\,w_2-1)(z\,w_1-1)}\right)}{(1+z^2)^2(z\,w_2-1)(z\,w_1-1)}+$$
$$+z^2\cdot\frac{z^2(1+w_1\,w_2)-z(w_2+w_1)+1}{(1+z^2)^2(z\,w_2-1)(z\,w_1-1)}$$

and also

$$Q(z) := \frac{z^4\,\mathrm{hypergeom}\left([1,1],[\,],-\frac{z^2\,(z-1)(1+z)}{(1+z^2)(z\,w_2-1)(z\,w_1-1)}\right)}{(1+z^2)^2(z\,w_2-1)(z\,w_1-1)}+\frac{z^2}{1+z^2}$$

The coefficient $c(n,j,k)$ of $z^n\,{w_1}^j\,{w_2}^k$ is obtained by straight expansion and avoiding paths are then enumerated by $C(n,j) = c(n,j,j)$. The corresponding formulae of the Theorem statement are obtained directly by symbolic expansions.

The computations are rather intensive and, for instance, the 4-variable GF that "lifts" $F(z,u,-1)$ is found to be

$$\frac{u\,z^2\left(1-z\,w_2-z\,w_1+z^2\,w_1\,w_2+z^2-z^3\,w_2-z^3\,w_1+z^4\,w_1\,w_2+u\,z^4\right)}{(1+z^2)\left(z^4\,w_1\,w_2+u\,z^4-z^3\,w_2-z^3\,w_1+z^2\,w_1\,w_2+z^2-u\,z^2-z\,w_2-z\,w_1+1\right)} \tag{5}$$

It is to be noted that computations have been performed with the help of the computer algebra packages `Combstruct` and `Gfun` that are dedicated to automating computations in combinatorial analysis and have been developed in the Maple system for symbolic computation. □

3 Average-Case Analysis for the Random Graph Model

We show now how to estimate the robustness to link failures in a random graph that obeys the $G_{n,p}$ model. An *avoiding pair* of length ℓ in a graph is an *unordered* pair of paths, each of length ℓ, with a common source and a common destination, that may share some nodes, but are edge disjoint. We have:

Theorem 3. *The mean number of avoiding pairs of length ℓ between a random source and a random destination in a random graph obeying the $G_{n,p}$ model is*

$$N_\ell(n,p) := \frac{p^{2\ell}}{2n(n-1)} \sum_{j=0}^{\ell} Q_{\ell+1,j} \binom{n}{l+1+j} (l+1+j)!$$

where the coefficients $Q_{n,j}$ are given by Theorem 2.

Proof. The coefficient $1/2$ corresponds to the fact that one takes unordered pairs of paths; the coefficient $1/(n(n-1))$ averages over all possible sources and destinations; the factor $p^{2\ell}$ provides the edge weighting corresponding to $G_{n,p}$; the arrangement numbers account for the number of ways to embed an avoiding path into a graph by choosing certain nodes and assigning them in some order to an avoiding path; the coefficients $Q_{\ell+1,j}$ provide the basic counting of avoiding paths that build up avoiding pairs. □

Note 1. Since the $G_{n,p}$ model implies isotropy, the quantity $N_\ell(n,p)$ is also the mean number of avoiding pairs between *any fixed* source and destination s,t.

Robustness. A short table of initial values of $N_\ell(n,p)$ follows:

$$N_2 = \tfrac{1}{2}(n-2)(n-3)p^4 \qquad N_3 = \tfrac{1}{2}(n-2)(n-3)^2(n-4)p^6$$
$$N_4 = \tfrac{1}{2}(n-1)(n-2)(n-3)(n-4)(n-5)^2p^8$$
$$N_5 = \tfrac{1}{2}(n-2)(n-3)(n-4)(n-5)^2(n^3-11n^2+25n+32)p^{10}$$

From developments in the previous section, the formulæ are computable in low polynomial time (as a function of ℓ). They make it possible to estimate the mean number of avoiding pairs in graphs of a given size for all reasonable values of n, p, ℓ. Take for instance a graph with $n = 10^5$ nodes and an edge probability $p = 5 \cdot 10^{-5}$. This corresponds to a mean node degree that is extremely close to 5, so that, on average, each node has 5 neighbors. Then the mean values are

$$N_2 = 3.1 \cdot 10^{-6},\ N_3 = 7.8 \cdot 10^{-5},\ N_4 = 1.9 \cdot 10^{-5},\ N_5 = 4.8 \cdot 10^{-4},\ N_6 = 1.2 \cdot 10^{-2},$$
$$N_7 = 0.30,\ N_8 = 7.6,\ N_9 = 190,\ N_{10} = 4763,\ N_{11} = 119062,\ N_{12} = 2.9 \cdot 10^7$$

Thus, in this example, one expects to have short and multiple connections between source and destination provided paths of length 8 are allowed. This numerical example also shows that there are rather sharp transitions. The formula of Theorem 3, that entails the following rough approximation

$$N_\ell(n,p) \approx \frac{1}{2}\, n^{2\ell-2} p^{2\ell} \tag{6}$$

precisely accounts for such a sharpness phenomenon.

In the introduction, we have defined ℓ-robustness as multiple connectivity by edge-disjoint paths of length *at most* ℓ. In fact, Equation 5 leads to explicit expressions for generalized avoiding pairs of type (ℓ_1, ℓ_2) that are made of two paths, of lengths ℓ_1, ℓ_2. It can then be seen that the bottleneck for existence of pairs (ℓ_1, ℓ_2) with $\ell_1, \ell_2 \leq \ell$ is in fact the case (ℓ, ℓ). Thus, since $N_\ell(n,p) \to 0$ when $\frac{p}{p_r(n,\ell)\to 0,}$ the function

$$p_r(n,\ell) = 2^{\frac{1}{2\ell}} n^{-1+\frac{1}{\ell}}$$

is a cut-off point for ℓ-robustness and an $(\leq \ell, \leq \ell)$-avoiding pair is expected or not depending on whether p/p_r tends to 0 or to ∞.

Corollary 2. *Any fixed pair in a $G_{n,p}$ graph is almost certainly not $\ell - robust$ if $p/p_r(n) \to 0$.*

Proof. When $\frac{p}{p_r(n,\ell)} \to 0$, then the expected number $N_\ell(n,p)$ of the desired pairs of paths tends to 0 and so does the probability of existence of at least one such pair of paths (since this probability, by Markov Inequality, is bounded from above by the expectation). Thus, with probability tending to 1, there is no pair of edge disjoint paths between the two vertices and these two vertices are, almost certainly, not $\ell - robust$. □

4 Thresholds in the Random Graph Model

In this section we provide bounds for the probability (and thus the threshold, if it exists) of the existence, between any fixed pair of vertices, of two edge-disjoint paths of length at most ℓ, by proving the following:

- We give an estimation of the value $p_L \equiv p_L(n,\ell)$ such that $G_{n,p}$ graphs with $p \leq p_L$ do not satisfy the desired property of the existence, between any fixed pair of nodes, of two edge-disjoint paths between some pair of vertices, with probability tending to 1 as n goes to infinity.
- We present a value $p_U \equiv p_U(n,\ell)$ such that almost every $G_{n,p}$ graph with $p \geq p_U$ has almost all its source-destination pairs of vertices connected by at least two edge-disjoint paths of length at most ℓ.

Theorem 4. *Define*

$$P_L(n,\ell) = \left(\log \frac{n^2}{\log n}\right)^{\frac{1}{\ell}} n^{-1+\frac{1}{\ell}}$$

Then, for $p \leq P_L(n,\ell)$, almost surely, there exists a pair of vertices in the $G_{n,p}$ graph that does not have the $\ell - robustness$ property.

Proof. By using the threshold function for diameter. □

Theorem 5. *Define*

$$P_U(n,\ell) = 2\left(\log\left(n^2 \log n\right)\right)^{\frac{1}{\ell}} n^{-1+\frac{1}{\ell}}$$

Then, for $p \geq P_U(n,\ell)$, almost surely, almost all pairs of vertices of a $G_{n,p}$ graph have the ℓ − robustness property.

Proof. Consider two independent distributions G_{n,p_1} and G_{n,p_2} on the same set of vertices. Let $E_i (i = 1, 2)$ be the events "G_{n,p_i} has diameter ℓ".

Consider the graph $\tilde{G}$ obtained when we superimpose an instance $G' \in G_{n,p_1}$ and an instance $G'' \in G_{n,p_2}$ and OR them (i.e., $\tilde{G}$ has an edge joining u, v iff at least one of G', G'' has). Clearly $\tilde{G} \in G_{n,p}$ with

$$p = p_1(1-p_2) + p_2(1-p_1) + p_1p_2 = p_1 + p_2 - p_1p_2$$

In fact, if u, v are joined in G' by a path P_1 and in G'' by a path P_2, then these two paths both exist in $\tilde{G}$. For p around the threshold for diameter ℓ of $G_{n,p}$ and $\ell = o(n)$, the number of pairs u, v of $\tilde{G}$ for which the paths of G', G'' overlap is $o(n^2)$, thus the vast majority of pairs of vertices ($n^2 - o(n^2)$ of them) in $\tilde{G}$ is connected via two edge disjoint paths of length $\leq \ell$.

This gives approximately a $p_U \leq p_1 + p_2 - p_1p_2$ and if $p_1 = p_2 = p_0^{(\ell)}$ (p_0 a threshold for diameter ℓ or $\ell + 1$) then

$$p_U \leq 2p_0^{(\ell)} - \left(p_0^{(\ell)}\right)^2 \leq 2(2\log n - \log c)^{\frac{1}{\ell}} n^{\frac{1}{\ell}-1}$$

from [4], where c can be adjusted so that the diameter is almost surely ℓ (see [4], Corollary 12, p. 237). □

5 A Discussion on the Extension to All Pairs

In this section we show how our results can be used to provide a tighter bound for p_U.

Lemma 1. *For every graph $G(V, E)$, if vertices u, v are each connected to a specific vertex $x \in V$ via two edge disjoint paths each of length ℓ, then u, v are connected in G via two edge disjoint paths, each of length at most 3ℓ.*

Proof. For simplicity, let the two (edge disjoint) paths from u to x be colored blue and the two (edge disjoint) paths from v to x be colored red. Take one of the two red paths and mark the first red-blue intersection vertex x_1 of it (it always exists such a vertex since it is in the worst case $x_1 = x$). Now take the other red path and mark the first red-blue intersection vertex x_2 (again this vertex can be x). There are two cases:
Case 1: Vertices x_1, x_2 are in different blue paths. Then the Lemma is easily proved by simply following the two different blue parts and then continuing with the two different red ones. Note that the two blue parts are edge disjoint,

the two red continuations are also edge disjoint and there is no red-blue edge. *Case 2:* Both x_1, x_2 are on the same blue path. Let x_1 the closest to u on this blue path. Take the first $u-v$ path to be from u (on this blue path) to x_1 and then from x_1 to v (by the same red path which defined x_1) and the second $u-v$ path be composed by the other red path from v to x_2, then the blue part from x_2 to x and then the unused other blue path returning to u. Again, there is obviously no edge intersection.

With respect to length, the worst case is clearly Case 2, where the second constructed path has pieces from three of the four initial paths, leading to length at most 3ℓ. □

Lemma 1 yields the following corollary:

Corollary 3. *For every graph $G(V,E)$ if there exists a vertex $x \in V$ such that for all vertices $u, v \in V (u, v \neq x)$ each of u, v connects to x via two edge disjoint paths of length at most ℓ, then the diameter of G is at most 3ℓ and each $u, v \in V$ is connected via two edge disjoint paths of length at most 3ℓ.*

Theorem 6. *Given $G_{n,p}$, if $p(n,\ell)$ is such that the probability that two specific nodes of G are connected via two edge disjoint paths of length at most ℓ is at least $1-\theta$ (where $\theta = o\left(\frac{1}{n}\right)$), then all pairs of nodes u, v of G are each connected via two edge disjoint paths of length at most 3ℓ with probability at least $1 - n\theta$.*

Proof. By applying to the probability of existence of paths between all pairs of vertices the fact that the probability of a union of events (existence of paths between two specific nodes) is bounded from above by the sum of the probabilities. □

Theorem 6 can provide an upper bound for the all pairs problem, by using an upper bound p_u such that for $p \geq p_u$, for every instance of $G_{n,p}$, any fixed (or random) pair has the $\ell - robustness$ property with probability tending to 1 as n tends to infinity. The derivation of such a bound could be approached by the computation of the Second Moment of the $\ell - robustness$ distribution, a computation that seems to be of major technical difficulty, and will be further examined in the future.

6 Conclusions and Further Research

We estimated here tightly and also asymptotically the mean value of the ways to get at least two edge disjoint paths between any two specific nodes of $G_{n,p}$ graphs. We pose as an open problem the calculation of the second moment for strengthening the threshold and also the extension of the problem to the existence of k edge disjoint paths.

References

1. N. Alon and J. Spencer, "The Probabilistic Method", John Wiley & Sons, 1992.
2. D. Bauer, F. Boesch, C. Suffel and R. Tindell, "Connectivity extremal problems and the design of reliable probabilistic networks", The Theory of Applications of Graphs, John Wiley and Sons, 1981.
3. F. T. Boesch, F. Harary and J. A. Cabell, "Graphs and models of communication networks vulnerability: connectivity and persistence", Networks, vol. 11, pp. 57–63, 1981.
4. B. Bollobás, "Random Graphs", Academic Press, 1985.
5. A. Broder, A. Frieze, S. Suen and E. Upfal, "Optimal construction of edge-disjoint paths in random graphs", SIAM Journal on Computing 28, 541-574. Also in the Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 603 - 612, 1994.
6. F. Chyzak, Ph. Flajolet and B. Salvy, "Studies in automatic Combinatorics, Volume II", INRIA 1998. Available at URL http://pauillac.inria.fr/algo/libraries/autocomb/autocomb.html
7. P. Erdös and A. Renyi, "On the evolution of random graphs", Magyar Tud. Akad. Math. Kut. Int. Kozl. 5, pp. 17–61, 1960.
8. Ph. Flajolet and R. Sedgewick, "An Introduction to the Analysis of Algorithms", Addison Wesley, 1996.
9. Ph. Flajolet and R. Sedgewick, "Analytic Combinatorics", book in preparation, 1999 (Individual chapters are available as INRIA Research Reports 1888, 2026, 2376, 2956, 3162).
10. J. Hromkovic, R. Klasing, E. Stoehr and H. Wagener, "Gossiping in Vertex-Disjoint Paths Mode in d-dimensional Grids and Planar Graphs", Proceedings of the 1st European Symposium on Algorithms (ESA), pp. 200-211, LNCS vol. 726, 1993.
11. S. Janson, D. Knuth, T. Łuczak and B. Pittel, "The Birth of the Giant Component", Random Structures and Algorithms, vol. 4, pp. 232-355, 1993.
12. Z. Kedem, K. Palem, P. Spirakis and M. Yung, "Faulty Random Graphs: reliable efficient-on-the-average network computing", Computer Technology Institute (CTI) Technical Report, 1993.
13. S. Nikoletseas, K. Palem, P. Spirakis, M. Young, "Short Vertex Disjoint Paths and Multiconnectivity in Random Graphs: Reliable Network Computing", 21st International Colloquium on Automata, Languages and Programming (ICALP), Jerusalem, pp. 508 – 515, 1994.
14. J. Spencer, "Ten Lectures on the Probabilistic Method", SIAM, 1987.

The $(\sigma + 1)$-Edge-Connectivity Augmentation Problem without Creating Multiple Edges of a Graph

Satoshi Taoka and Toshimasa Watanabe

Department of Circuits and Systems, Faculty of Engineering,
Hiroshima University
1-4-1, Kagamiyama, Higashi-Hiroshima, 739-8527 Japan
Phone: +81-824-24-7666 (Taoka), -7662 (Watanabe)
Facsimile: +81-824-22-7028
E-mail : {taoka,watanabe}@infonets.hiroshima-u.ac.jp

Abstract. The unweighted *k-edge-connectivity augmentation problem* (kECA for short) is defined by "Given a σ-edge-connected graph $G = (V, E)$, find an edge set E' of minimum cardinality such that $G' = (V, E \cup E')$ is $(\sigma+\delta)$-edge-connected and $\sigma+\delta = k$", where E' is called a *solution* to the problem. Let kECA(S,SA) denote kECA such that both G and G' are simple.
The subject of the present paper is $(\sigma + 1)$ECA(S,SA) (or kECA(S,SA) with $k = \sigma + 1$). Let $\mathcal{M}$ be any maximum matching of a certain graph $R(G)$ whose vertex set V_R consists of vertices representing all leaves of G. From $\mathcal{M}$ we obtain an edge set E'_0, with $|E'_0| = |\mathcal{M}|$, such that each edge connects vertices in distinct leaves of G. Let $\mathcal{L}_1$ be the set of leaves to be created by adding E'_0 to G, and $\mathcal{K}_1$ the set of remaining leaves of G.
The main result is to propose two $O(\sigma^2|V|\log(|V|/\sigma)+|E|+|V_R|^2)$ time algorithms for finding the following solutions: (1) an optimum solution if G has at least $2\sigma + 6$ leaves or if $|\mathcal{L}_1| \leq |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves; (2) a $\frac{3}{2}$-approximate solution if $|\mathcal{L}_1| > |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves.

1 Introduction

The unweighted *k-edge-connectivity augmentation problem* (kECA for short) is described as follows: "Given a σ-edge-connected graph $G = (V, E)$, find an edge set E' of minimum cardinality such that $G' = (V, E \cup E')$ is $(\sigma + \delta)$-edge-connected and $\sigma + \delta = k$." We often denote G' as $G + E'$, and E' is called a *solution* to the problem. Let kECA(*,**) denote kECA with the following restriction (i) and (ii) on G and E', respectively: (i) * is set to S if G is required to be simple, and * is left to mean that G may be a multiple graph; (ii) ** is set to MA if creation of new multiple edges in constructing G' is allowed, and is set to SA otherwise. In kECA(*,SA), if G is simple then so is G', or if G has multiple edges then any multiple edge of G' exists in G. As for kECA,

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 169–185, 2000.
© Springer-Verlag Berlin Heidelberg 2000

kECA(*,MA) has mainly been discussed so far. See [3,5,7,8,12,13,20,21,22,23] for the results. It is natural for us to assume that $|V| \geq \sigma + 2$ in $(\sigma+1)$ECA(S,SA): in $(\sigma+1)$ECA(*,SA), we may have $|V| \leq \sigma + 1$.

As related results, kECA(S,SA) for G having no edges was first discussed in [6], where the problem that is more general than kECA(S,SA) is considered. An $O(|V|+|E|)$ algorithm for 2ECA(S,SA) can be obtained by slightly modifying the one given in [3] for 2ECA(*,MA). As for 3ECA(*,SA), [23] proposed an $O(|V|+|E|)$ algorithm for 3ECA(*,MA), and showed that if $|V| \geq 4$ then this algorithm finds an optimum solution to 3ECA(*,SA). Concerning $(\sigma+1)$ECA(S,SA) with $|V| \geq \sigma + 2$ for $\sigma \in \{3, 4\}$, [15] proposed an $O(|V| \log |V| + |E|)$ algorithm. Other related results have been reported in [14,16]. T. Jordán showed in [10] that kECA(S,SA) is NP-hard in general, and [2] proposed an $O(|V|^4)$ algorithm for kECA(S,SA) for any fixed k.

The subject of the present paper is $(\sigma+1)$ECA(S,SA), that is, kECA(S,SA) with $k = \sigma + 1$. Let $\mathcal{M}$ be any maximum matching of the *leaf-graph* $R(G)$ whose vertex set V_R consists of vertices representing all leaves of G. (The definition of $R(G)$ is going to be given later). From $\mathcal{M}$ we obtain a certain edge set E_0', with $|E_0'| = |\mathcal{M}|$, such that each edge connects vertices in distinct leaves of G. Let $\mathcal{L}_1$ be the set of leaves to be created by adding E_0' to G, and $\mathcal{K}_1$ the set of remaining leaves of G.

The main result of the paper is to propose two $O(\sigma^2|V| \log(|V|/\sigma) + |E| + |V_R|^2)$ time algorithms for finding the following solutions for $(\sigma+1)$ECA(S,SA):

(1) an optimum solution if G has at least $2\sigma + 6$ leaves or if $|\mathcal{L}_1| \leq |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves;
(2) a $\frac{3}{2}$-approximate solution if $|\mathcal{L}_1| > |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves.

A central concept in solving kECA is a *t-edge-connected component* of G: a maximal set of vertices such that G has at least t edge-disjoint paths between any pair of vertices in the set [22]. A t-edge-connected component whose degree (the number of edges connecting vertices in the set to those outside of it) is equal to the edge-connectivity of G is called a *leaf*. Although $(\sigma+1)$ECA(S,SA) can be solved almost similarly to general kECA(*,MA), the only difference is that the augmenting step has to choose a pair of leaves, each containing a vertex such that they are not adjacent in G. (Such a pair of leaves is called a *nonadjacent pair*.) This requires addition of some other characteristics or processes in finding solutions by means of structural graphs: a structural graph is introduced in [11], and is used as a useful tool that reduces time complexity in finding a solution to kECA(*,MA) in [7,13].

This paper adopts the operation, called *edge-interchange*, in finding a solution, where it was introduced in [20,21] in order to reduce time complexity of [22]. A set of two nonadjacent pairs of leaves is called a *D-combination* if they are disjoint. The augmenting step in solving $(\sigma+1)$ECA(S,SA) repeats both choosing a nonadjacent pair of leaves and enlarging a $(\sigma+1)$-edge-connected component by means of edge-interchange (or an analogous operation). Hence

obtaining an optimum solution requires finding a maximum set of nonadjacent pairs of leaves such that any two members in the set form a D-combination and, therefore, this is reduced to finding a maximum matching of the leaf-graph $R(G)$ of G. The point of $(\sigma+1)$ECA(S,SA) is that a solution E' is closely related to a maximum matching $\mathcal{M}$ of $R(G)$.

The paper is organized as follows. Basic definitions and several basic results on σ-edge-connected componets and leaf-graphs are given in Section 2. In Section 3, results on maximum matchings of leaf-graphs are briefly mentioned. Edge-interchange operation is explained in Section 4. Section 5 discusses $(\sigma+1)$ECA(S,SA) when G has less than $2\sigma+6$ leaves, and Section 6 considers $(\sigma+1)$ECA(S,SA) when G has at least $2\sigma+6$ leaves.

All proofs are omitted becase of space limitation.

2 Preliminaries

2.1 Basic Definitions

Technical terms not specified here can be identified in [1,4,9,19]. An *undirected graph* $G=(V(G),E(G))$ consists of a finite and nonempty set of vertices $V(G)$ and a finite set of undirected edges $E(G)$, where $V(G)$ and $E(G)$ are often denoted as V and E, respectively. An edge e incident upon two vertices u,v in G is denoted by $e=(u,v)$ unless any confusion arises. We denote $V(e)=\{u,v\}$, or generally $V(K)=\{u,v\in V|(u,v)\in K\}$ for a subset $K\subseteq E$. For disjoint sets X, $X'\subset V$, we denote $(X,X';G)=\{(u,v)\in E|u\in X \text{ and } v\in X'\}$, where it is often written as (X,X') if G is clear from the context. We denote $d_G(X)=|(X,\overline{X};G)|$. This is called the *degree* of X (in G). We set $d_G(S)=0$ if $S=\emptyset$. If $X=\{v\}$ then $d_G(\{v\})$ is denoted simply as $d_G(v)$ and is the total number of edges (v,v'), $v'\neq v$, incident upon v. We often denote $d_G(S)$ as $d(S)$ if G is clear from the context. A path between vertices u and v is often called a (u,v)-*path* and denoted by $P_G(u,v)$, and is often written as $P(u,v)$ if G is clear from the context. For two vertices u, v of G, let $\lambda(u,v;G)$, or simply $\lambda(u,v)$, denote the maximum number of pairwise edge-disjoint paths between u and v.

For a set $X\subseteq V$, let $G[X]$ denote the subgraph having X as its vertex set and $\{(u,v)\in E|u,v\in X\}$ as its edge set. $G[X]$ is called the *subgraph* of G *induced* by X (or the *induced subgraph* of G by X). *Deletion* of $X\subseteq V$ from G is to construct $G[V-X]$, which is often denoted as $G-X$. If $X=\{v\}$ then we often denote $G-v$ for simplicity. Deletion of $Q\subseteq E$ from G defines a spanning subgraph of G, denoted by $G-Q$, having $E-Q$ as its edge set. If $Q=\{e\}$ then we denote $G-e$. For a set E' of edges such that $E'\cap E=\emptyset$, let $G+E'$ denote the graph $(V,E\cup E')$. If $E'=\{e\}$ then we denote $G+e$.

Let $K\subseteq E$ be any minimal set such that $G-K$ has more components than G. K is called a *separator* of G, or in particular a (X,Y)-separator if any vertex of X and any one of Y are disconnected in $G-K$. If $X=\{u\}$ or $Y=\{v\}$ then it is denoted as a (u,Y)-separator or a (X,v)-separator, respectively. A *minimum* (X,Y)-*separator* K of G is a (X,Y)-separator of minimum cardinality. Such

K is often called an (X, Y)-cut or an $|K|$-cut. It is known that a (u, v)-cut K has $|K| = \lambda(u, v; G)$. A *minimum separator* K of G is a separator of minimum cardinality among all separators of G, and $|K|$ is called the *edge-connectivity* (denoted by σ) of G; particularly we call such $K \subseteq E$ a *minimum cut* (of G). G is said to be *k-edge-connected* if $\lambda(G) \geq k$. A *k-edge-connected component* (k-component, for short) of G is a subset $S \subseteq V$ satisfying the following (a) and (b): (a) $\lambda(u, v; G) \geq k$ for any pair $u, v \in S$; (b) S is a maximal set that satisfies (a). Let $\Gamma_G(k)$ denote the set of all k-components of G. In a graph G with $\lambda(G) = \sigma$, a $(\sigma + 1)$-component S with $d_G(S) = \sigma$ is called a *leaf* $(\sigma + 1)$-*component* of G (or a leaf of G, for short). It is known that $\lambda(G) \geq k$ if and only if V is a k-component. Note that distinct k-components are disjoint sets. Each 1-component is often called a *component*.

Note that we assume that $|V| \geq \sigma + 2$ in $(\sigma + 1)$ECA(S,SA), the subject of the paper.

A *cactus* is an undirected connected graph in which any pair of cycles share at most one vertex. A *structural graph* $F(G)$ of G with $\lambda(G) = \sigma$ is a representation of all minimum cuts of G and is introduced in [11]. We use the term "nodes of $F(G)$" to distinguish them from vertices of G. $F(G)$ is an edge-weighted cactus of $O(|V|)$ nodes and edges such that each tree edge (an edge which is a bridge in $F(G)$) has weight $\lambda(G)$ and each cycle edge (an edge included in any cycle) has weight $\lambda(G)/2$. Let $F(G)$ be a structural graph of G. Particularly if σ is odd then $F(G)$ is a weighted tree. (Examples of G and $F(G)$ will be given in Figs. 1 and 2.) Each vertex in G maps to exactly one node in $F(G)$, and $F(G)$ may have some other nodes, call *empty nodes*, to which no vertices of G are mapped. Let $\epsilon(G) \subseteq V(F(G))$ denote the set of all empty nodes of $F(G)$. Note that any minimum cut of G is represented as either a tree edge or a pair of two cycle edges in the same cycle of $F(G)$, and vice versa. Let ρ: $V \to V(F(G)) - \epsilon(G)$ denote this mapping. We use the following notations: $\rho(X) = \{\rho(v) | v \in X\}$ for $X \subseteq V$, and $\rho^{-1}(Y) = \{v \in V | \rho(v) \in Y\}$ for $Y \subseteq V(F(G))$. $\rho(\{v\})$ or $\rho^{-1}(\{v\})$ is written as $\rho(v)$ or $\rho^{-1}(v)$, respectively, for notational simplicity. For any cut $(X, V(F(G)) - X; F(G))$, if summation of weights of all edges contained in the cut is equal to σ then $(\rho^{-1}(X), V - \rho^{-1}(X); G)$ is a σ-cut of G. Note that the cut of $F(G)$ consists of either one tree edge or a pair of two cycle edges in the same cycle of $F(G)$. Conversely, for any σ-cut $(X, V - X; G)$, $F(G)$ has at least one cut $(Y, V(F(G)) - Y; G)$ in which summation of weight of all edges contained in the cut is equal to σ, where Y is a node set of $F(G)$ such that $\rho(X) = Y - \epsilon(G)$. Each $(\sigma + 1)$-component S of G is represented as a vertex $\rho(S) \in V(F(G)) - \epsilon(G)$ in $F(G)$, and, for any vertex $v \in V(F(G)) - \epsilon(G)$, $\rho^{-1}(v)$ is a $(\sigma + 1)$-component of G. For $v \in V(F(G))$, if summation of weights of all edges that are incident to v in $F(G)$ equals to σ, then v is called a *leaf node* (that is a degree-1 vertex in a tree or a degree-2 vertex in a cycle). Note that, for any leaf node v, $\rho^{-1}(v)$ is a leaf of G, conversely, for any leaf L of G, $\rho(L)$ is a leaf node of $F(G)$. It is shown that $F(G)$ can be constructed in $O(|V||E|)$ time [11] or in $O(\sigma^2|V| \log(|V|/\sigma) + |E|)$ time [7].

Two edges e_1, e_2 are said to be *independent* if and only if $V(e_1) \cap V(e_2) = \emptyset$, and a set $Q \subseteq E$ is called an *independent set* or a *matching* of G if and only if any pair of edges in Q are independent. An independent set of maximum cardinality in G is called a *maximum matching* of G.

Proposition 1. *[5] For distinct sets $X, Y \subset V$ of any graph $G = (V, E)$,*

$$d(X) + d(Y) = d(X - Y) + d(Y - X) + 2|(V - X \cup Y, X \cap Y)|, \tag{2.1}$$

$$d(X) + d(Y) = d(X \cap Y) + d(X \cup Y) + 2|(X - Y, Y - X)|. \tag{2.2}$$

Let $\lceil x \rceil$ ($\lfloor x \rfloor$, respectively) denote the minimum integer no smaller (the maximum one no greater) than x.

2.2 σ-Components and Leaf-Graphs

Let $\lambda(G) = \sigma > 0$. Let X_1, X_2 be distinct $(\sigma+1)$-components of G. The pair $\{X_1, X_2\}$ are called an *adjacent pair* (denoted as $X_1 \chi X_2$) if any two vertices $w \in X_1$ and $w' \in X_2$ are adjacent in G, or called a *nonadjacent pair* (denoted as $X_1 \overline{\chi} X_2$) otherwise. Let

$$V_C = \{v | v \text{ represents an individual } (\sigma+1)\text{-component of } G\}$$

and let $S(v) \in \Gamma_G(\sigma+1)$ denote the one represented by $v \in V_C$. Let $C(G) = (V_C, E_C)$ be defined by V_C and $E_C = \{(v, v') | v, v' \in V_C \text{ and } S(v) \overline{\chi} S(v')\}$, and it is called the *component graph* of G. Let $LF(G) = \{X \in \Gamma_G(\sigma+1) | X \text{ is a leaf of } G\}$ and $V_R = \{v | v \text{ represents an individual leaf of } G\} \subseteq V_C$. Let $Y(v)$ denote the leaf $(\sigma+1)$-component represented by $v \in V_R$. Let $R(G) = (V_R, E_R)$ be the subgraph of $C(G)$ defined by $E_R = \{(v, v') \in E_C | v, v' \in V_R \text{ and } Y(v) \overline{\chi} Y(v')\}$, and it is called the *leaf-graph* of G.

Property 1. $R(G)$ is simple.

Let Y_i, $i = 1, 2, 3, 4$, be distinct leaves of G. A set of two nonadjacent pairs $\{Y_1, Y_2\}, \{Y_3, Y_4\}$ is called a *D-combination* if they are disjoint (that is, $\{Y_1, Y_2\} \cap \{Y_3, Y_4\} = \emptyset$). In general, for $2t$ distinct leaves Y_i, $i = 1, \ldots, 2t$, of G with $t \geq 2$, a set of t nonadjacent pairs $\{Y_1, Y_2\}, \ldots, \{Y_{2t-1}, Y_{2t}\}$ is called a *D-set* of G if any two pairs of the set form a D-combination. Let $Y_1 \chi \{Y_2, Y_3\}$ denote that both $Y_1 \chi Y_2$ and $Y_1 \chi Y_3$ hold. A D-combination $\{\{Y_1, Y_2\}, \{Y_3, Y_4\}\}$ is called an *I-combination* (denoted as $\{Y_1, Y_2\} \angle \{Y_3, Y_4\}$) if either $Y_1 \chi \{Y_3, Y_4\}$ or $Y_2 \chi \{Y_3, Y_4\}$ holds. If neither $\{Y_1, Y_2\} \angle \{Y_3, Y_4\}$ nor $\{Y_3, Y_4\} \angle \{Y_1, Y_2\}$ holds then we denote $\{Y_1, Y_2\} \not\angle \{Y_3, Y_4\}$.

We first show some basic results on $R(G)$ and leaves of G.

Proposition 2. *Suppose that G is simple. Then either $|Y| = 1$ or $|Y| \geq \sigma + 2$ for any $Y \in LF(G)$.*

Since each leaf Y has $d_G(Y) = \sigma$, we obtain the next proposition by Proposition 2.

Proposition 3. *Suppose that G is simple. If $\{Y_1, Y_2\} \subseteq LF(G)$ is an adjacent pair then $|Y_1| = |Y_2| = 1$.*

Proposition 4. *$d_{R(G)}(v) \geq \max\{|V_R| - (\sigma + 1), 0\}$ for any $v \in V_R$.*

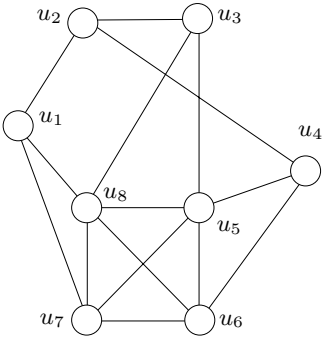


Fig. 1. A simple graph G with $\lambda(G) = 3$ and $|LF(G)| = 4$.

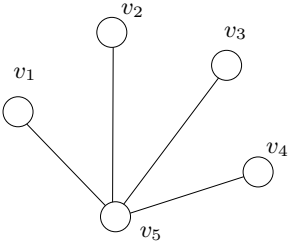


Fig. 2. A structural graph $F(G)$ of G in Fig. 1, where all edge-weights are 3 and none of them are written. In this case leaves Y_i in $LF(G)$ of the graph G shown in Fig. 1 are represented as nodes v_i of $F(G)$ for $i = 1, \ldots, 5$: it may happen that G has a node to which no corresponding leaf of $LF(G)$ exists.

2.3 Examples

Let $G = (V, E)$ with $|V| \geq \sigma + 2$ and $\lambda(G) = \sigma$ be any given simple graph. Let $OPT(M)$ or $OPT(S)$ denote the cardinality of an optimum solution to $(\sigma+1)$ECA(*,MA) or to $(\sigma+1)$ECA(S,SA) for G, respectively. For $\sigma = 3$, we give an example such that $OPT(S) = OPT(M)+1$. For the graph G with $|LF(G)| = 4$ shown Fig. 1, $R(G)$ is given in Fig. 3. The set of edges $\{(u_1, u_3), (u_2, u_4)\}$ is an optimum solution to 4ECA(*,MA), while $\{(u_1, u_3), (u_2, u_8), (u_3, u_7)\}$ is an optimum solution to 4ECA(S,SA) and, therefore, $OPT(S) = 3 = OPT(M) + 1$.

3 Maximum Matchings of Leaf-Graphs

One of requirements in finding a solution to $(\sigma+1)$ECA(S,SA) or $(\sigma+1)$ECA(*, SA) with $\sigma \geq 1$ is to obtain a largest D-set. Hence, in this section, the cardinality of a maximum D-set is investigated by considering a maximum matching $\mathcal{M}$ of $R(G)$.

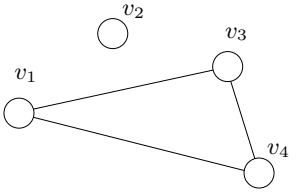


Fig. 3. The leaf-graph $R(G)$ of G in Fig. 1.

Let $\mathcal{M}$ denote any fixed maximum matching of $R(G)$ in the following discussion unless otherwise stated, where we assume that $\lambda(G) = \sigma \geq 1$.

Proposition 5. *$|\mathcal{M}|$ satisfies one of the following (1)–(3).*

(1) If $|V_R| \geq 2\sigma + 1$ or if σ is even and $|V_R| = 2\sigma$ then $|\mathcal{M}| = \lfloor |V_R|/2 \rfloor$.
(2) If σ is odd and $|V_R| = 2\sigma$ then

$$\lfloor |V_R|/2| \rfloor - 1 \leq |\mathcal{M}| \leq \lfloor |V_R|/2 \rfloor.$$

(3) If $|V_R| \leq 2\sigma - 1$ then

$$\max\{0, \min\{|V_R| - \sigma, \lfloor |V_R|/2 \rfloor\}\} \leq |\mathcal{M}| \leq \lfloor |V_R|/2 \rfloor.$$

Corollary 1. *Suppose that $|V_R| = 2\sigma$ and $\sigma = 2m + 1$. If $|\mathcal{M}| = \lfloor |V_R|/2 \rfloor - 1$ then $G = (V, E)$ is a complete bipartite graph with $V = X \cup Y$, $X \cap Y = \emptyset$, $|X| = |Y| = \sigma$ and $E = \{(x, y) | x \in X, y \in Y\}$.*

The relationship among G, $C(G)$ and $R(G)$ shows the following proposition concerning $|V_R|$, $|\mathcal{M}|$ and $|E'|$ of any optimum solution E' to $(\sigma+1)$ECA(S,SA).

Proposition 6. *Let E' be any solution to G in $(\sigma+1)$ECA(S,SA) and $\mathcal{M}$ be a maximum matching of $R(G)$. Then*

$$|V_R| - |\mathcal{M}| \leq |E'|. \tag{3.1}$$

4 Augmentation by Edge-Interchange

We explain an operation called edge-interchange which was originally introduced in [20,21] for an efficient augmentation. It is also used in [14,15,16,17,18]. Let $LF(G) = \{Y_1, \ldots, Y_q\}$ $(q = |LF(G)|)$ denote the class of all leaves of G and choose $y_i \in Y_i$ as the representative of Y_i. Let

$$Y(G) = \{y_i | Y_i \in LF(G)\}, \quad q \geq 2, \text{ and } r = \lceil q/2 \rceil.$$

We can easily prove the next proposition.

Proposition 7. *If there is a set E' of edges, each connecting vertices of G, such that $E' \cap E = \emptyset$ and $V(E') = Y(G) \subseteq S$ for some $(\sigma+1)$-component S of $G + E'$, then $S = V$.*

Let Y stand for $Y(G)$ in the rest of the section.

4.1 Attachments

We have $d_G(Y_i) = \sigma$ and $\lambda(y_i, y_j; G) = \sigma$ for any $y_i, y_j \in Y$ $(i \neq j)$. An edge set F is called an *attachment* (for G) if and only if the following (1) through (4) hold:

(1) $V(F) \subseteq Y$,
(2) $F \cap E(G) = \emptyset$,
(3) $V(e) \neq V(e')$ $(\forall e, e' \in F, e \neq e')$, and
(4) if q $(= |LF(G)|)$ is odd then F has at most one pair f, f' such that $|V(f) \cap V(f')| = 1$; or if q is even then F has no such pair.

Let F be any attachment for G. For each $e = (u, v) \in F$, $G + F$ has a new $(\sigma + 1)$-component, denoted by $\mathcal{A}(e, G + F)$, containing $V(e)$.

We are going to show that we can find a minimum attachment $Z(\sigma + 1) = \{e_1, \ldots, e_r\}$ $(r = \lceil q/2 \rceil)$ such that $\lambda(G + Z(\sigma + 1)) = \sigma + 1$. Although there are two cases: $r = 1$ and $r \geq 2$, we discuss only the latter case in the following. (Note that if $r = 1$ then we immediately obtain the desired attachment F.)

4.2 Finding a Minimum Attachment

Suppose that there are an attachment F for G and vertices $y_{ij} \in Y - V(F)$, $1 \leq i, j \leq 2$, where y_{11}, y_{12}, y_{21} are distinct, and if y_{22} is equal to one of the other three then we assume that $y_{22} = y_{21}$ (see Fig. 4). We use the following

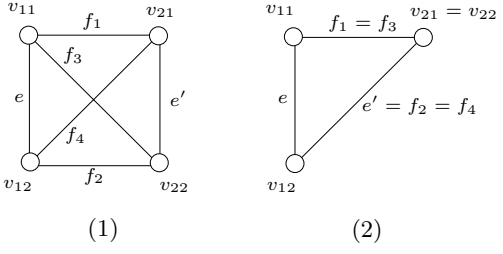


Fig. 4. The edges e, e' and f_i, $1 \leq i \leq 4$: (1) $y_{21} \neq y_{22}$; (2) $y_{21} = y_{22}$.

notations:

$$L = G + F, \quad e = (y_{11}, y_{12}), \quad e' = \begin{cases} (y_{21}, y_{22}) & \text{if } y_{21} \neq y_{22} \\ (y_{12}, y_{21}) & \text{if } y_{21} = y_{22}, \end{cases}$$

$$\mathcal{A}(e) = \mathcal{A}(e, L + \{e, e'\}), \quad \mathcal{A}(e') = \mathcal{A}(e', L + \{e, e'\}),$$

$$f_1 = (y_{11}, y_{21}), \quad f_2 = (y_{12}, y_{22}), \quad f_3 = (y_{11}, y_{22}), \quad f_4 = (y_{12}, y_{21}),$$

where we set $f_1 = f_3$ and $e' = f_2 = f_4$ if $y_{21} = y_{22}$, and

$$\mathcal{A}(f_i) = \begin{cases} \mathcal{A}(f_i, L + \{f_1, f_2\}) & \text{if } 1 \le i \le 2 \\ \mathcal{A}(f_i, L + \{f_3, f_4\}) & \text{if } 3 \le i \le 4. \end{cases}$$

Note that $e, e', f_i \notin E(L), 1 \le i \le 4$. We have the following two cases.

Case I: $\mathcal{A}(e) \cap \mathcal{A}(e') = \emptyset$; Case II: $\mathcal{A}(e) \cap \mathcal{A}(e') \neq \emptyset$ (that is, $\mathcal{A}(e) = \mathcal{A}(e')$).

For Case I, we are going to show that there are two edges f, f', with $V(f) \cup V(f') = V(e) \cup V(e')$, such that

$$\mathcal{A}(e) \cup \mathcal{A}(e') \subseteq \mathcal{A}(f, L + \{f, f'\}) = \mathcal{A}(f', L + \{f, f'\}).$$

That is, we can add two edges so that one $(\sigma+1)$-component containing $\mathcal{A}(e) \cup \mathcal{A}(e')$ may be obtained. Finding and adding such a pair of edges f, f' is called *edge-interchange* (with respect to $V(e_1) \cup V(e_2)$).

Suppose that $\mathcal{A}(e) \cap \mathcal{A}(e') = \emptyset$. Note that $y_{21} \neq y_{22}$ in this case. Let K be any fixed $(\mathcal{A}(e), \mathcal{A}(e'))$-cut of $L + \{e, e'\}$, and let B_i, $1 \le i \le 2$, denote the two sets of vertices in $L + \{e, e'\}$ such that $B_1 \cup B_2 = V$, $B_2 = V - B_1$, $K = (B_1, B_2; L + \{e, e'\})$, $\mathcal{A}(e) \subseteq B_1$ and $\mathcal{A}(e') \subseteq B_2$. $|K| = \sigma = \lambda(y_1, y_2; L'')$ for any $y_i \in B_i$, $1 \le i \le 2$, where L'' denotes L, $L + e$, $L + e'$ or $L + \{e, e'\}$. K is a (y_1, y_2)-cut of L. Suppose that f and f' satisfy either (i) or (ii):

(i) $f = f_1$, $f' = f_2$, or (ii) $f = f_3$, $f' = f_4$,
where $\{f, f'\} \cap E(L) = \emptyset$.

The next proposition shows a property of edge-interchange.

Proposition 8. *If $\mathcal{A}(e) \cap \mathcal{A}(e') = \mathcal{A}(f_1) \cap \mathcal{A}(f_2) = \emptyset$ then $\mathcal{A}(f_3) \cap \mathcal{A}(f_4) \neq \emptyset$, that is, $\mathcal{A}(f_3) = \mathcal{A}(f_4)$.*

Let $\{f, f'\}$ denote the following pair of edges:

$\{e, e'\}$ if $\mathcal{A}(e) = \mathcal{A}(e')$ (the case with $V(e) \cap V(e') = \emptyset$ is included);

$\{f_1, f_2\}$ if $\mathcal{A}(e) \cap \mathcal{A}(e') = \emptyset$ and $\mathcal{A}(f_1) = \mathcal{A}(f_2)$;

$\{f_3, f_4\}$ if $\mathcal{A}(e) \cap \mathcal{A}(e') = \mathcal{A}(f_1) \cap \mathcal{A}(f_2) = \emptyset$.

Clearly, $\{f, f'\} \cap E(L) = \emptyset$. Such a pair f, f' are called an *augmenting pair* (with respect to $\{y_{11}, y_{12}, y_{21}, y_{22}\}$) of L.

Corollary 2. *Let $L' = L + \{f, f'\}$ for any augmenting pair f, f'. Then $L' - f'$ has no σ-cut separating $V(f')$ from $V(f)$. That is, if $L' - f'$ has a σ-cut K separating a vertex of $V(f')$ from $V(f)$ then K separates the two vertices of $V(f')$.*

From Corollary 2, other important properties (Proposition 9–11) of edge-interchange are obtained.

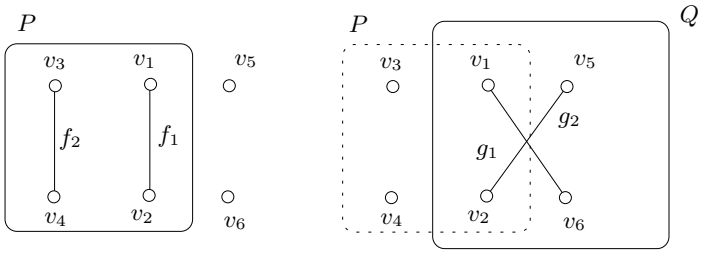


Fig. 5. The two (σ+1)-components $\mathcal{A}(f_1, G+\{f_1, f_2\})$ and $\mathcal{A}(g_1, G+\{g_1, g_2\})$ produced by two augmenting pairs $\{f_1, f_2\}$ and $\{g_1, g_2\}$, respectively.

Proposition 9. *Suppose that G has six leaves $Y_i \in LF(G)$ $(1 \leq i \leq 6)$, and choose $y_i \in Y_i$ as a representative of each Y_i. Suppose that $\{f_1, f_2\}$ is an augmenting pair with respect to $\{y_i | 1 \leq i \leq 4\}$ of G. If $\mathcal{A}(f_1, G + \{f_1, f_2\})$ is a leaf then, for each $i \in \{1, 2\}$, there is an augmenting pair $\{g_1, g_2\}$ with respect to $V(f_i) \cup \{y_5, y_6\}$ of G such that $\mathcal{A}(g_1, G + \{g_1, g_2\})$ is not a leaf (see Fig. 5).*

By Proposition 9, we obtain the following procedure that is a modified version of the procedure given in [15]. It finds a sequence of edges $e_1, \ldots, e_r$ $(r = \lceil |LF(G)|/2 \rceil \geq 1)$ by repeating edge-interchange operation, where handling the case with $|LF(G)| = 2$ is included. Note that edges with which we are concerned are those connecting vertices belonging to distinct leaves. If an edge g connects a vertex in a leaf Y_i and another vertex in a leaf Y_j $(i \neq j)$ then, for simplicity, we say that g *connects* Y_i *and* Y_j.

Procedure *FIND_EDGES;*
begin
1. $G_1 \leftarrow G$; $\pi \leftarrow LF(G)$; $i \leftarrow 1$; $E'_1 \leftarrow \emptyset$;
2. **while** $\pi \neq \emptyset$ **do**
begin
3. **if** $|\pi| = 2$ **then**
4. $f_i \leftarrow$ an edge connecting the two leaves of π; $E''_i \leftarrow \{f_i\}$;
5. **else if** $|\pi| \leq 5$ **then**
6. Find an augmenting pair $E''_i = \{f_i, f'_i\}$ by Proposition 8;
7. **else** /* $|\pi| \geq 6$ */
8. Find an augmenting pair $E''_i = \{f_i, f'_i\}$ by Proposition 9;
9. $E'_{i+1} \leftarrow E'_i \cup E''_i$; $G_{i+1} \leftarrow G_i + E''_i$; $\pi \leftarrow \pi - \{Y(v) | v \in V(E''_i)\}$; $i \leftarrow i + 1$;
end
end;

Proposition 10. *G_{i+1} has a leaf containing $\mathcal{A}(f_i, G_{i+1})$ if and only if $|LF(G_i)| = 5$ just after the execution of Step 9 in FIND_EDGES.*

Note that executing Step 6 or Step 8 once can be done in $O(|V_R|)$ by using a structural graph $F(G)$, and we can construct $F(G)$ in $O(\sigma^2|V|\log(|V|/\sigma)+|E|)$ time (see [7]). The details are omitted here.

The next proposition holds for the edge set E' produced by *FIND_EDGES*.

Proposition 11. *Let $Z(\sigma+1)=\{e_1,\ldots,e_r\}$ $(r=\lfloor|LF(G)/2\rfloor)$ be given by FIND_EDGES. Then $Z(\sigma+1)$ is a minimum attachment such that $\lambda(G')=\sigma+1$, where $G'=G+Z(\sigma+1)$. Furthermore the procedure runs in $O(\sigma^2|V|\log(|V|/\sigma)+|E|+|V_R|^2)$ time.*

5 $(\sigma+1)$ECA(S,SA) for G Having Less Than $2\sigma+6$ Leaves

We denote $LF(G)=\{Y_i|1\le i\le q\}$ $(q=|LF(G)|)$, $Y(G)=\{y_1,\ldots,y_q\}$ and $V_R=\{v_1,\ldots,v_q\}$, where each y_i is represented as v_i in $R(G)$. First we consider the case where G has two or three leaves.

Proposition 12. *If $q=2$ then the following (1) or (2) holds.*

(1) If $Y_1\overline{\chi}Y_2$ then $|\mathcal{M}|=1$, there are two vertices $y_i\in Y_i$, $i=1,2$, such that $E'=\{(y_1,y_2)\}$ is a solution, and $OPT(S)=OPT(M)=1$.

(2) If $Y_1\chi Y_2$ then $|\mathcal{M}|=0$, there are three vertices $y_i\in Y_i$ $(i=1,2)$, $x\in V-(Y_1\cup Y_2)$ such that $E'=\{(y_1,x),(y_2,x)\}$ is a solution, and $OPT(S)=2=OPT(M)+1$.

Proposition 13. *If $q=3$ and there exist two leaves Y_1, Y_2 with $Y_1\overline{\chi}Y_2$ then $|\mathcal{M}|=1$, there are distinct edges e_1,e_2 such that $E'=\{e_1,e_2\}$ is a solution, and $OPT(S)=OPT(M)=2$.*

Next we consider the remaining case where $3\le q<2\sigma+6$. For each $e'=(x',y')\in\mathcal{M}$, we can choose two vertices $x\in Y(x')$, $y\in Y(y')$, and let $e=(x,y)$ be an edge, which is not included in E. We fix such an edge e for each $e'\in\mathcal{M}$, and let

$$E'_0=\{e=(x,y)\mid(x',y')\in\mathcal{M}\}.$$

Proposition 14. *$|E'_0|=|\mathcal{M}|$ and $E'_0\cap E=\emptyset$.*

In the rest of this section, we consider the graph $G+E'_0$. First we define two sets $\mathcal{L}_1$ and $\mathcal{K}_1$ as follows.

Let $G_1=G+E'_0$ and let $\mathcal{L}_1$ be the set of new leaves of G_1 created by adding E'_0 to G. Clearly $|\mathcal{L}_1|\le|\mathcal{M}|$. Let $\mathcal{K}_1=LF(G+E'_0)-\mathcal{L}_1$ $(\subseteq LF(G))$. Since $\mathcal{M}$ is a maximum matching of $R(G)$, Proposition 3 shows that each leaf in $\mathcal{K}_1$ consists of only one vertex and that the set of vertices $\mathcal{K}'_1=\{x\mid\{x\}\in\mathcal{K}_1\}$ induces a complete graph of G and of $G+E'_0$.

We are going to propose an $O(\sigma^2|V|\log(|V|/\sigma)+|E|+|V_R|^2)$ time algorithm such that it finds an optimum solution if $|\mathcal{L}_1|\le|\mathcal{K}_1|$ and such that a $\frac{3}{2}$-approximate solution if $|\mathcal{L}_1|>|\mathcal{K}_1|$. Note that we have $|\mathcal{L}_1|\le|\mathcal{K}_1|$ if $|\mathcal{M}|\le\lfloor|V_R|/3\rfloor$.

Proposition 15. *Let* $\{y'_1\}, \{y'_2\} \in \mathcal{K}_1$ $(y'_1 \neq y'_2)$ *and* $Y_1, Y_2 \in \mathcal{L}_1$ $(Y_1 \neq Y_2)$. *If* $\{(y_1, y'_1), (y_2, y'_2)\}$ *is not an augmenting pair with* $y_1 \in Y_1$ *and* $y_2 \in Y_2$ *then there are* $y_3 \in Y_1$ *and* $y_4 \in Y_2$ *such that* $\{(y_4, y'_1), (y_3, y'_2)\}$ *is an augmenting pair and* $(y_4, y'_1), (y_3, y'_2) \notin E$ *(See Fig. 6).*

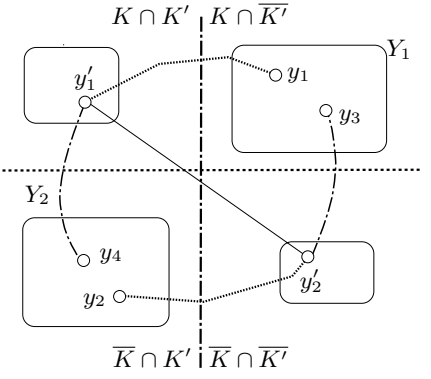


Fig. 6. A situation for Proposition 15

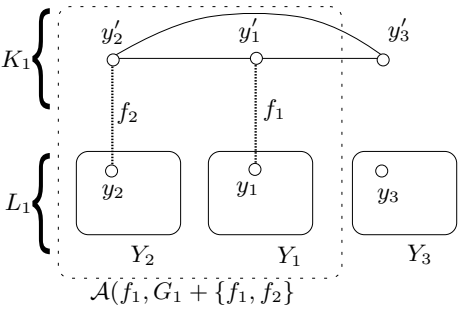


Fig. 7. $\mathcal{A}(f_1, G + \{f_1, f_2\})$ in the proof of Proposition 16

We obtain the next proposition by Propositions 9 and 15.

Proposition 16. *Assume that* $|\mathcal{L}_1| \geq 3$ *and* $|\mathcal{K}_1| \geq 3$. *Then there exists an augmenting pair* $\{f_1, f_2\}$ *such that* $f_1 = (y_1, y'_1) \notin E \cup E'_0$, $f_2 = (y_2, y'_2) \notin E \cup E'_0$, $\{\{y'_1\}, \{y'_2\}\} \subseteq \mathcal{K}_1$ $(y'_1 \neq y'_2)$, $\mathcal{L}_1$ *has two distinct sets* Y_1, Y_2 *with* $y_1 \in Y_1$, $y_2 \in Y_2$ *and* $\mathcal{A}(f_1, G + \{f_1, f_2\})$ *is not a leaf. Furthermore* $\mathcal{L}_1 \cup \mathcal{K}_1 - \{\{y'_1\}, \{y'_2\}\}, Y_1, Y_2\}$ *is the set of all leaves in* $G_1 + \{f_1, f_2\}$. *(See Fig. 7)*

Next we are going to discuss the case where $|\mathcal{L}_1| \leq 2$ or $|\mathcal{K}_1| \leq 2$.

Proposition 17. *Suppose that* $|\mathcal{L}_1| \leq 2$ *and* $|\mathcal{L}_1| \leq |\mathcal{K}_1|$. *Then there exists a set* $E'_2 = \{f_1, \ldots, f_{|\mathcal{K}_1|}\}$ *such that* $\lambda(G_1 + E'_2) \geq \sigma + 1$ *and* $E'_2 \cap (E \cup E'_0) = \emptyset$.

It remains to consider the cases ($|\mathcal{L}_1| \geq 3$ and $|\mathcal{K}_1| \leq 2$) and ($|\mathcal{L}_1| \leq 2$ and $|\mathcal{L}_1| > |\mathcal{K}_1|$), for which the next proposition holds.

Proposition 18. *Suppose that one of the following (1)–(3) holds: (1)* $|\mathcal{L}_1| \geq 3$ *and* $|\mathcal{K}_1| \leq 2$*; (2)* $|\mathcal{L}_1| = 2$ *and* $|\mathcal{K}_1| = 1$*; (3)* $|\mathcal{L}_1| = 2$ *and* $|\mathcal{K}_1| = 0$. *Let* $q_1 = |LF(G_1)|$ *and* $r_1 = \lceil \frac{q_1}{2} \rceil$. *Then there exists a set* $E''_2 = \{f_1, \ldots, f_{r_1}\}$ *such that* $\lambda(G_1 + E''_2) \geq \sigma + 1$ *and* $E''_2 \cap (E \cup E'_0) = \emptyset$.

The discussion from Propositions 16 through 18 is summarized in the following procedure *FIND_EDGES2*.

Procedure *FIND_EDGES2;*
begin
1. $G_0 \leftarrow G$; $\pi \leftarrow LF(G)$; $E'_0 \leftarrow \emptyset$; $\rho \leftarrow \emptyset$;
2. Find an edge set E'_0 as in Proposition 14; $G_1 \leftarrow G_0 + E'_0$;
   Determine $\mathcal{L}_1$ and $\mathcal{K}_1$; $i \leftarrow 1$;
3. **while** $\mathcal{K}_i \neq \emptyset$ **do**
   begin
4. **if** $|\mathcal{L}_i| \geq 3$ and $|\mathcal{K}_i| \geq 3$ **then**
   Find an augmenting pair $\{f, f'\}$ by Proposition 16, $E''_i \leftarrow \{f, f'\}$;
5. **else if** $|\mathcal{L}_i| \leq 2$ and $|\mathcal{L}_i| \leq |\mathcal{K}_i|$ **then**
   Find an edge set E''_i by Proposition 17;
6. **else**
   Find an edge set E''_i by Proposition 18;
7. Construct $\mathcal{K}_{i+1}$ and $\mathcal{L}_{i+1}$; $E'_i \leftarrow E'_{i-1} \cup E''_i$; $G_{i+1} \leftarrow G_i + E''_i$; $i \leftarrow i+1$;
   end;
8. **if** $\lambda(G_i) = \sigma$ **then**/* the case with $|\mathcal{L}_i| \neq 0$ */
   Find an edge set E''_i by Proposition 18; $E'_{i+1} \leftarrow E'_{i-1} \cup E''_i$;
end;

Proposition 19. *FIND_EDGES2 produces an optimum solution if* $|\mathcal{L}_1| \leq |\mathcal{K}_1|$.

Proposition 20. *FIND_EDGES2 gives a* $\frac{3}{2}$*-approximate solution if* $|\mathcal{L}_1| > |\mathcal{K}_1|$.

Remark 1. Let $\mathcal{M}$ be any maximum matching of $R(G)$. If $|\mathcal{M}| \leq \lfloor \frac{|LF(G)|}{3} \rfloor$ then $|\mathcal{L}_1| \leq |\mathcal{K}_1|$ and we can find an optimum solution in polynomial time. If $\lfloor \frac{|LF(G)|}{3} \rfloor < |\mathcal{M}| \leq \lfloor \frac{|LF(G)|}{2} \rfloor$ then $|\mathcal{L}_1| \leq |\mathcal{K}_1|$ or $|\mathcal{L}_1| > |\mathcal{K}_1|$. Since the proof of NP-completeness of kECA(S,SA) in [10] is given for the case with $|\mathcal{M}| = \lfloor \frac{|LF(G)|}{2} \rfloor$, we consider approximate solutions if $|\mathcal{L}_1| > |\mathcal{K}_1|$.

Theorem 1. *Suppose that* $|LF(G)| \leq 2\sigma+6$. *Then FIND_EDGES2 can find an optimum solution if* $|\mathcal{L}_1| \leq |\mathcal{K}_1|$, *or a* $\frac{3}{2}$*-approximate solution if* $|\mathcal{L}_1| > |\mathcal{K}_1|$, *in* $O(\sigma^2|V|\log(|V|/\sigma) + |E|)$ *time.*

6 $(\sigma+1)$ECA(S,SA) for G Having at Least $2\sigma+6$ Leaves

In this case, Proposition 5(3) shows that any maximum matching $\mathcal{M}$ of $R(G)$ has $|\mathcal{M}| = \lfloor \frac{|LF(G)|}{2} \rfloor$. First, some basic results on nonadjacent pairs and edge interchange operation are going to be given.

Proposition 21. *Suppose that there are a nonadjacent pair of leaves* $Y_1, Y_2 \in LF(G)$ *and two vertices* $y_i \in Y_i$, $i = 1, 2$, *with* $(y_1, y_2) \notin E$, *such that* $G' = G + \{(y_1, y_2)\}$ *has a leaf* S *containing* $Y_1 \cup Y_2$. *Let* $\mathcal{L}' = \{Y \subseteq S | Y \in \Gamma_G(\sigma+1)\}$, $X = Y_1 \cup Y_2$ *and* $Z = \bigcup_{Y \in LF(G)-\{Y_1,Y_2\}} Y$. *Then* $|(X, Z; G)| \leq \sigma - 1$ *if* $|\mathcal{L}'| \geq 3$.

The next proposition can be proved by using Propositon 21.

Proposition 22. *Suppose $\sigma \geq 3$ and let $\mathcal{M}' = \{(v_{2i-1}, v_{2i}) | 1 \leq i \leq m\} \subseteq \mathcal{M}$ for some $m \leq |\mathcal{M}|$, and put $Y_j = Y(v_j)$ for each $v_j \in V_R$.*

(1) If $|\mathcal{M}'| \geq 2$ and there are distinct indices i, j with $1 \leq i, j \leq m$ such that $\{Y_{2i-1}, Y_{2i}\} \not\parallel \{Y_{2j-1}, Y_{2j}\}$ then (i) and (ii) hold.

(i) These leaves are partitioned into a D-combination $\{\{L'_1, L'_2\}, \{L'_3, L'_4\}\}$ having four vertices $y_t \in L'_t$, $t = 1, 2, 3, 4$, such that $G + \{(y_1, y_2), (y_3, y_4)\}$ has a $(\sigma+1)$-component S containing all L'_t, $t = 1, 2, 3, 4$.

(ii) The $(\sigma+1)$-component S' of $G + \{(y_1, y_2)\}$ such that $L'_1 \cup L'_2 \subseteq S'$ is not a leaf.

(2) If $|\mathcal{M}'| \geq \lceil \sigma/2 \rceil + 1$ and no such pair of indices as in (1) exist then, for each $(v_{2i-1}, v_{2i}) \in \mathcal{M}'$, there are vertices $y_{2i-1} \in Y_{2i-1}$ and $y_{2i} \in Y_{2i}$ such that $G' = G + \{(y_{2i-1}, y_{2i})\}$ is a simple graph having a $(\sigma+1)$-component X which is not a leaf and which contains $Y_{2i-1} \cup Y_{2i}$.

Proposition 23. *Suppose that there is a set $\mathcal{M}' = \{(v_{2i-1}, v_{2i}) | 1 \leq i \leq m\} \subseteq \mathcal{M}$ for some m with $\sigma + 2 \leq m \leq |\mathcal{M}|$, and put $Y_i = Y(v_i)$ for each $v_i \in V_R$. Then there is an edge $(v_{2h-1}, v_{2h}) \in \mathcal{M}'$ with $\{Y_1, Y_2\} \not\parallel \{Y_{2h-1}, Y_{2h}\}$.*

By combining Propositions 9, 22 and 23, we obtain the following proposition.

Proposition 24. *Suppose that there is a set $\mathcal{M}' = \{f_i = (v_{2i-1}, v_{2i}) | 1 \leq i \leq m\} \subseteq \mathcal{M}$ for some m with $\sigma+3 \leq m \leq |\mathcal{M}|$, and put $Y_i = Y(v_i)$ for each $v_i \in V_R$. Then there exists an augmenting pair $\{e'_1, e'_2\}$ with respect to $Y_1, Y_2, Y_{2j-1}, Y_{2j}$ such that $G + \{e'_1, e'_2\}$ is simple and has no leaf S with $Y_1 \cup Y_2 \cup Y_{2j-1} \cup Y_{2j} \subseteq S$, where $\{f_1, f_j\} \subseteq \mathcal{M}'$.*

Based on Proposition 24, the next procedure *FIND_EDGES3* is obtained.

Procedure *FIND_EDGES3;*
begin
1. $G_1 \leftarrow G$; $\pi \leftarrow LF(G)$; $i \leftarrow 1$; $E'_0 \leftarrow \emptyset$;
2. **while** $\pi \neq \emptyset$ **do**
begin
3. **if** $|\pi| \leq 3$ **then**
4. Find an edge set E''_i as E'in Proposition 12(1) or 13;
5. **else**
begin /* $|\pi| \geq 4$ */
6. Find a matching $\mathcal{M}'' = \{(v_{2p-1}, v_{2p}) | 1 \leq p \leq m'\}$ of $R(G_i)$,
where if $|\pi| \leq 2\sigma + 6$ then $m' \leftarrow \lfloor \pi/2 \rfloor$, otherwise $m' \leftarrow \sigma + 3$;
7. **if** $|\pi| \leq 2\sigma + 6$ **then**
begin
Choose $E'_s \subseteq E'_i$ with $|E'_s| = \sigma + 3 - m'$ appropriately;
$\mathcal{M}' \leftarrow \mathcal{M}'' \cup \{(v, w) \in E_R | (v', w') \in E'_s, v' \in Y(v), w' \in Y(w)\}$;

/* $\mathcal{M}'$ is a matching on $R(G)$ in the case.*/
end;
else
$\mathcal{M}' \leftarrow \mathcal{M}''$;
8. Find an augmenting pair E_i'' as $\{e_1', e_2'\}$ in Proposition 24
by choosing $f_1 \in \mathcal{M}''$; /* Note that $|\mathcal{M}'| = \sigma + 3$. */
9. **if** $f_j \in \mathcal{M}' - \mathcal{M}''$ for f_j of Proposition 24 **then**
begin /* In the case with $|\pi| \leq 2\sigma + 6$ */
$E_i' \leftarrow E_i' - \{(y_{2j-1}, y_{2j})\}$, $G_i \leftarrow G_i - \{(y_{2j-1}, y_{2j})\}$, where
$y_{2j-1} \in Y_{2j-1}$ and $y_{2j} \in Y_{2j}$;
end;
10. $E_{i+1}' \leftarrow E_i' \cup E_i''$; $G_{i+1} \leftarrow G_i + E_i''$;
$\pi \leftarrow \pi - \{Y(v) | v \in V(E_i'')\}$; $i \leftarrow i + 1$;
end;
end;

Proposition 25. *Any set final E_i' obtained at the termination of FIND_EDGES3 is a minimum attachment such that $\lambda(G') = \sigma + 1$, where $G' = G + E'$.*

Theorem 2. *If G has at least $2\sigma + 6$ leaves then the algorithm FIND_EDGES3 correctly finds a solution E' to $(\sigma+1)$ECA(S,SA) for any given G with $\lambda(G) = \sigma$ in $O(\sigma^2|V|\log(|V|/\sigma) + |E| + |V_R|^2)$ time.*

7 Concluding Remarks

The paper has proposed

(1) an $O(\sigma^2|V|\log(|V|/\sigma) + |E| + |V_R|^2)$ time algorithm for finding an optimum solution if G has at least $2\sigma + 6$ leaves or if $|\mathcal{L}_1| \leq |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves,
(2) an $O(\sigma^2|V|\log(|V|/\sigma) + |E|)$ time one for a $\frac{3}{2}$-approximate solution if $|\mathcal{L}_1| > |\mathcal{K}_1|$ and G has less than $2\sigma + 6$ leaves.

We can improve the first algorithm to an $O(\sigma^2|V|\log(|V|/\sigma) + |E|)$ time one by devising how to check whether or not $\{f_1, f_2\}$ is an augmenting pair, and whether or not $\mathcal{A}(f_1, G + \{f_1, f_2\})$ is a leaf in Proposition 9.

Acknowledgments

The research of T.Watanabe is partly supported by the Grant in Aid for Scientific Research on Priority Areas of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No.10205219.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
2. J. Bang-Jensen and T. Jordán, *Edge-connectivity augmentation preserving simplicity*, SIAM J. Discrete Math., 11 (1998), pp. 603–623.
3. K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653–655.
4. S. Even, *Graph Algorithms*, Pitman, London, 1979.
5. A. Frank, *Augmenting graphs to meet edge connectivity requirements*, SIAM J. Discrete Mathematics, 5 (1992), pp. 25–53.
6. H. Frank and W. Chou, *Connectivity considerations in the design of survivable networks*, IEEE Trans. Circuit Theory, CT-17 (1970), pp. 486–490.
7. H. N. Gabow, *Applications of a poset representation to edge connectivity and graph rigidity*, in Proc. 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 812–821.
8. ———, *Efficient splitting off algorithms for graphs*, in Proc. 26th ACM Symposium on Theory of Computing, 1994, pp. 696–705.
9. T. C. Hu, *Integer Programming and Network Flows*, Addison-Wesley, Reading, Mass, 1969.
10. T. Jordán, *Two NP-complete augmentation problems*, Tech. Rep. PP-1997-08, Odense University, Denmark, March 1997. http://www.imada.ou.dk/Research/Preprints/j-l.html.
11. A. V. Karzanov and E. A. Timofeev, *Efficient algorithm for finding all minimal edge cuts of a nonoriented graph*, Cybernetics, (1986), pp. 156–162. Translated from Kibernetika, 2 (1986), 8–12.
12. H. Nagamochi and T. Ibaraki, *A faster edge splitting algorithm in multigraphs and its application to the edge-connectivity augmentation problem*, Tech. Rep. 94017, Kyoto University, 1994.
13. D. Naor, D. Gusfield, and C. Martel, *A fast algorithm for optimally increasing the edge connectivity*, SIAM J. Comput., 26 (1997), pp. 1139–1165.
14. D. Takafuji, S. Taoka, and T. Watanabe, *Simplicity-preserving augmentation to 4-edge-connect a graph*, IPSJ SIG Notes, AL-33–5 (1993), pp. 33–40.
15. S. Taoka, D. Takafuji, and T. Watanabe, *Simplicity-preserving augmentation of the edge-connectivity of a graph*, Tech. Rep. of IEICE of Japan, COMP93–73 (1994), pp. 49–56.
16. S. Taoka and T. Watanabe, *Efficient algorithms for the edge-connectivity augmentation problem of graphs without increasing edge-multiplicity*, IPSJ SIG Notes, AL-42-1 (1994), pp. 1–8.
17. ———, *Minimum augmentation to k-edge-connect specified vertices of a graph*, in Lecture Notes in Computer Science 834(D-Z du and X-S Zhang(Eds.): Algorithms and Computation), Springer-Verlag, Berlin, 1994, pp. 217–225. (Proc. 5th International Symposium on Algorithms and Computation(ISAAC'94)).
18. ———, *Smallest augmentation to k-edge-connect all specified vertices in a graph*, IPSJ SIG Notes, AL-38–3 (1994), pp. 17–24.
19. R. E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, PA, 1983.

20. T. Watanabe, *An efficient way for edge-connectivity augmentation*, Tec. Rep. ACT-76-UILU-ENG-87-2221, Coordinated Science Lab., University of Illinois at Urbana, Urbana, IL 61801, April 1987. Also presented at Eighteenth Southeastern International Conference on Combinatorics, Graph Theory, Computing, No.15, Boca Raton, FL, U.S.A., February 1987.
21. ——, *A simple improvement on edge-connectivity augmentation*, Tech. Rep., IEICE of Japan, CAS87–203 (1987), pp. 43–48.
22. T. Watanabe and A. Nakamura, *Edge-connectivity augmentation problems*, J. Comput. System Sci., 35 (1987), pp. 96–144.
23. T. Watanabe and M. Yamakado, *A linear time algorithm for smallest augmentation to 3-edge-connect a graph*, IEICE Trans. Fundamentals of Japan, E76-A (1993), pp. 518–531.

On the Hardness of Approximating Some NP-Optimization Problems Related to Minimum Linear Ordering Problem
(Extended Abstract)

Sounaka Mishra and Kripasindhu Sikdar

Stat-Math Unit, Indian Statistical Institute, Calcutta 700 035
{res9513,sikdar}@isical.ac.in

Abstract. We study hardness of approximating several minimaximal and maximinimal NP-optimization problems related to the minimum linear ordering problem (MINLOP). MINLOP is to find a minimum weight acyclic tournament in a given arc-weighted complete digraph. MINLOP is APX-hard but its unweighted version is polynomial time solvable. We prove that, MIN-MAX-SUBDAG problem, which is a generalization of MINLOP, and requires to find a minimum cardinality maximal acyclic subdigraph of a given digraph, is, however APX-hard. Using results of Håstad concerning hardness of approximating independence number of a graph we then prove similar results concerning MAX-MIN-VC (respectively, MAX-MIN-FVS) which requires to find a maximum cardinality minimal vertex cover in a given graph (respectively, a maximum cardinality minimal feedback vertex set in a given digraph). We also prove APX-hardness of these and several related problems on various degree bounded graphs and digraphs.

Keywords : NP-optimization problems, Minimaximal and maximinimal NP-optimization problems, Approximation algorithms, Hardness of approximation, APX-hardness, L-reduction.

1 Introduction

In this paper we deal with hardness of approximating several minimum-maximal and maximum-minimal NP-complete optimization problems on graphs as well as related maximum/minimum problems. In general, for any given instance x of such a problem, it is required to find a minimum (respectively, maximum) weight (or, cardinality) maximal (respectively, minimal) feasible solution with respect to a partial order on the set of feasible solutions of x. The terminology of minimaximal and maximinimal is apparently first used by Peters et.al. [18], though the concept has received attention of many others, specially in connection with many graph problems. For example, minimum chromatic number and its maximum version, the achromatic number [11,4]; maximum independent set and minimaximal independent set (minimum independent dominating set) [12,13];

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 186–199, 2000.
© Springer-Verlag Berlin Heidelberg 2000

minimum vertex cover and maximinimal vertex cover [17,19]; minimum dominating set and maximinimal dominating set [16,5] and a recent systematic study of minimaximal and maximinimal optimization problems by Manlove [19].

We are led to investigation of several such graph problems while considering a generalization of the minimum linear ordering problem (MINLOP). Given a complete digraph $G_n = (V, A_n)$ on a set $V = \{v_1, v_2, \ldots v_n\}$ of n vertices with nonnegative integral arc weights, the MINLOP is to find an acyclic tournament [10] on V with minimum total arc weight. This is a known NP-complete optimization problem [9] and some results about approximation solutions and hardness of approximability of MINLOP have been obtained in [20]. Two problems related to MINLOP are the maximum acyclic subdigraph (MAX-SUBDAG) and the minimum feedback arc set (MIN-FAS) problems. Given a digraph $G = (V, A)$, the MAX-SUBDAG (respectively, MIN-FAS) problem is to find a subset of $B \subseteq A$ of maximum (respectively, minimum) cardinality such that (V, B) (respectively, $(V, A - B)$) is an acyclic subdigraph (SUBDAG) of G. While MAX-SUBDAG is APX-complete [17] and has a trivial $\frac{1}{2}$-approximation algorithm, MIN-FAS is not known to be in APX, though it is APX-hard [14].

A generalization of MINLOP can be formulated as follows. Note that an acyclic tournament on V is indeed a maximal SUBDAG of G_n (i.e., a SUBDAG of G_n which is not contained in any SUBDAG of G_n). Thus we generalize MINLOP as the minimum weight maximal SUBDAG (MIN-W-MAX-SUBDAG) problem which requires to find a maximal SUBDAG of minimum total arc weight in any given arc weighted digraph (which is not necessarily a complete digraph). MIN-W-MAX-SUBDAG is thus APX-hard as its special case MINLOP is so. We show that unweighted version (i.e., all arc weights 1) of MIN-W-MAX-SUBDAG, called MIN-MAX-SUBDAG, is APX-hard even though MINLOP with constant arc weight is solvable in polynomial time.

The complementary problem of MIN-MAX-SUBDAG is the maximum cardinality minimal feedback arc set (MAX-MIN-FAS) in which it is required to find a minimal feedback arc set of maximum cardinality in a given digraph. The vertex version of this is the maximum cardinality minimal feedback vertex set (MAX-MIN-FVS). An NP-optimization problem related to MAX-MIN-FVS is MAX-MIN-VC, in which it is required to find a minimal vertex cover of maximum cardinality in a given graph. Another related problem is the minimum maximal independent set (MIN-MAX-IS) problem, where one is required to find a maximal IS (or an independent dominating set) of minimum cardinality for any given graph.

Since the decision versions of these optimization problems are NP-complete, it is not possible to find optimal solutions in polynomial time, unless P=NP. So a practical alternative is to find near optimal (or approximate) solutions in polynomial time. However, it is not always possible to obtain such solutions having desired approximation properties [2,12,15,17]. Thus it is of considerable theoretical and practical interest to provide some qualitative explanation for this by establishing results about hardness of obtaining such approximate solutions.

In this paper, we shall establish several results about hardness of approximating such problems using the standard technique of reduction of one problem to another. Due to restriction on the number of pages, we shall give outlines of most of the lengthy proofs, details of which are in [21]. The paper is organized as follows. In Section 2, we recall the relevant concepts about graphs, digraphs, and approximation algorithms. In Section 3, we first prove APX-hardness of MIN-MAX-SUBDAG for arbitrary digraph by reducing MAX-SUBDAG to it. Then, using the results of Håstad concerning hardness of approximating MAX-IS, we prove similar results about MAX-MIN-VC for arbitrary graphs and about MAX-MIN-FVS for arbitrary digraphs. In Section 4, we prove APX-hardness of MIN-FAS and MAX-SUBDAG for k-total-regular digraphs, for all $k \geq 4$. Then we show that MIN-MAX-SUBDAG is APX-hard for digraphs of maximum total degree 12. We also prove that MAX-MIN-VC is k-approximable for all graphs without any isolated vertex and having maximum degree k, $k \geq 1$, MAX-MIN-VC is APX-complete for all graphs of maximum degree 5, and MAX-MIN-FVS is APX-hard for all digraphs of maximum total degree 10. In Section 5, we show that, MIN-FVS is APX-complete for 6-regular graphs and MAX-MIN-FVS is APX-hard for all graphs of maximum degree 9. Finally, in Section 6, we make some concluding remarks.

2 Basic Concepts

We will denote a graph (i.e. an undirected graph) by $G = (V, E)$ and a digraph (i.e. a directed graph) by $G = (V, A)$, where $V = \{v_1, v_2, \ldots v_n\}$, E is the edge set and A is the arc set. An edge between vertices v_i and v_j will be denoted by $\{v_i, v_j\}$, whereas an arc from v_i to v_j will be denoted by the ordered pair (v_i, v_j). In an undirected graph G, *degree* of a vertex v_i is denoted as $d(v_i)$ which is the number of edges incident on v_i in G, and G is called *k-regular* if each vertex in G has degree k. In a digraph G, $d^+(v_i)$ and $d^-(v_i)$ are the number of arcs in G having v_i as the initial vertex and terminal vertex, respectively, and $d(v_i)$, the *total degree* of v_i is defined as $d(v_i) = d^+(v_i) + d^-(v_i)$. A digraph G is *k-total-regular* if for each vertex v_i, $d(v_i) = k$. A *path* $P(v_1, v_t)$ in $G = (V, E)$ (respectively, *dipath* in $G = (V, A)$) is a sequence of distinct vertices $(v_1, v_2, \ldots, v_t)$ such that $\{v_i, v_{i+1}\} \in E$ (respectively, $(v_i, v_{i+1}) \in A$) for $1 \leq i < t$. A path (respectively, dipath) $P(v_1, v_t)$ is called a *cycle* (respectively, *dicycle*) if $v_1 = v_t$.

A *feedback arc set* (FAS) (respectively a *directed acyclic subgraph* (SUBDAG)) in a digraph $G = (V, A)$ is an arc set $B \subseteq A$ such that the subdigraph $(V, A - B)$ (respectively (V, B)) is acyclic. Given a digraph $G = (V, A)$, a *minimal FAS* (respectively *maximal SUBDAG*) is an FAS (respectively SUBDAG) $B \subseteq A$ which does not contain (respectively is not contained in) another FAS (respectively SUBDAG). Given a graph $G = (V, E)$, $C \subseteq V$ is called a *vertex cover* (VC) if for each edge $\{v_i, v_j\} \in E$, C contains either v_i or v_j. A VC C is called a *minimal VC* of G if no proper subset of C is also a VC of G. $S \subseteq V$ is called an *feedback vertex set* (FVS) of G if the subgraph/subdigraph $G[V - S]$ induced by the vertex set $V - S$ is acyclic. Similarly a *minimal FVS* of G is defined.

The precise formulation of the problems considered in this paper are as follows:
MAX-SUBDAG (respectively, MIN-FAS)
Instance - A digraph $G = (V, A)$.
Solution - A SUBDAG (V, B) (respectively, an FAS B) of G.
Cost - $m(x, B) = |B|$.
Goal - max (respectively, min).

MAX-SUBDAG-k (respectively, MIN-FAS-k) is the problem of MAX-SUBDAG (respectively, MIN-FAS) on k-total-regular digraphs.

MIN-W-FVS
Instance - A pair $x = (G, w)$ where G is a graph/digraph and w assigns a non-negative integer to each $v \in V$.
Solution - An FVS F of G.
Cost - $m(x, F) = \sum_{v \in F} w(v)$.
Goal - min.

MIN-FVS is the unweighted version of MIN-W-FVS, i.e. MIN-W-FAS with $w(v) = 1$ for each $v \in V$. MIN-FVS-k is the problem of MIN-FVS on k-regular (respectively k-total-regular) graphs (respectively digraphs).

MIN-MAX-SUBDAG (respectively, MAX-MIN-FAS)
Instance - Same as that of MAX-SUBDAG.
Solution - A maximal SUBDAG (V, B) (respectively, a minimal FAS B) of G.
Cost - $m(x, B) = |B|$.
Goal - min (respectively, max).

MIN-MAX-SUBDAG$\leq k$ (respectively, MAX-MIN-FAS$\leq k$) is the problem of MIN-MAX-SUBDAG (respectively, MAX-MIN-FAS) on digraphs of total degree at most k.

MAX-MIN-FVS
Instance - Same as that of MIN-FVS.
Solution - A minimal FVS B of G.
Cost - $m(x, B) = |B|$.
Goal - max.

MAX-MIN-FVS-k is the problem of MAX-MIN-FVS on k-regular (respectively k-total-regular) graphs (respectively digraphs) and MAX-MIN-FVS$\leq k$ is the problem of MAX-MIN-FVS on graphs (respectively digraphs) of degree (respectively total-degree) at most k.

MAX-MIN-VC
Instance - A graph $x = G = (V, E)$.
Solution - A minimal VC C of G.
Cost - $m(x, C) = |C|$.
Goal - max.

MAX-MIN-VC$\leq k$ is the problem of MAX-MIN-VC on graphs of degree at most k.

Given an instance x of an NP optimization problem π and $y \in sol(x)$, the performance ratio of y with respect to x is defined by $R_\pi(x,y) = \max\{\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\}$ where $m^*(x)$ is the optimum value.

A polynomial time algorithm A for an NP optimization problem π is called an ϵ-*approximate* algorithm for π for some $\epsilon > 1$ if $R_\pi(x, A(x)) \leq \epsilon$ for any instance x of π, where $A(x)$ is the solution for x given by A. The class APX is the set of all NP optimization problems which have some ϵ-approximate algorithm.

An approximation algorithm A for an NP optimization problem π approximates the optimal cost within a factor of $f(n)$ if, for all instances x of π, it produces a solution $A(x)$ in polynomial time such that $R_\pi(x, A(x)) \leq f(|x|)$.

Among the approximation preserving reductions L-reduction [17] is the easiest one to use. π_1 is said to be *L-reducible* to π_2 [17], in symbols $\pi_1 \leq_L \pi_2$, if there exist two functions f, g and two positive constants α, β such that:

1. For any $x \in I_{\pi_1}$, $f(x) \in I_{\pi_2}$ is computable in polynomial time.
2. For any $x \in I_{\pi_1}$ and for any $y \in sol_{\pi_2}(f(x))$, $g(x,y) \in sol_{\pi_1}(x)$ is computable in polynomial time.
3. $m^*_{\pi_2}(f(x)) \leq \alpha \cdot m^*_{\pi_1}(x)$.
4. For any $x \in I_{\pi_1}$ and for any $y \in sol_{\pi_2}(f(x))$,
$|m^*_{\pi_1}(x) - m_{\pi_1}(x, g(x,y))| \leq \beta \cdot |m^*_{\pi_2}(f(x)) - m_{\pi_2}(f(x), y)|$.

We shall be using in this paper only the L-reduction though the hardness (or completeness) in the class APX is defined in terms of PTAS-reduction ($\leq_{PTAS}$) [6,2]. An NP optimization problem π is APX-hard if, for any $\pi' \in$ APX, $\pi' \leq_{PTAS} \pi$, and problem π is APX-complete if π is APX-hard and $\pi \in$ APX. However it is well known that [7,15] for any two NP optimization problems π_1 and π_2, if $\pi_1 \leq_L \pi_2$ and $\pi_1 \in$ APX, then $\pi_1 \leq_{PTAS} \pi_2$.

3 Hardness Results for Arbitrary Graphs/Digraphs

As already noted, MIN-W-MAX-SUBDAG is APX-hard, we now show that its unweighted version MIN-MAX-SUBDAG is also APX-hard, even though the unweighted version of MINLOP is solvable in polynomial time. For this it is enough to prove the following theorem, as MAX-SUBDAG is APX-complete [17].

Theorem 1. *MAX-SUBDAG* $\leq_L$ *MIN-MAX-SUBDAG with* $\alpha = 5$ *and* $\beta = 1$.

Proof. (Outline) For each instance $x = G = (V, A)$ of MAX-SUBDAG, we construct in polynomial time an instance $f(x) = G' = (V', A')$ of MIN-MAX-SUBDAG and with each feasible solution (V', S') of $f(x)$, we associate a feasible solution $g(S') = S = S' \cap A$ of x such that f and g satisfy the conditions of L-reduction with $\alpha = 5$ and $\beta = 1$.

Let $K = \{(v_i, v_j) | (v_i, v_j) \in A \text{ and } (v_j, v_i) \notin A\}$. For each arc $(v_i, v_j) \in K$, we introduce a new vertex v_{ij} for the construction of G'. Construct $G' = (V', A')$ as

follows: $V' = V \cup \{v_{ij} | (v_i, v_j) \in K\}$ and $A' = A \cup \{(v_j, v_i), (v_j, v_{ij}), (v_{ij}, v_j) | (v_i, v_j) \in K\}$. For an example, see Figure 1. Let $k = |K|$ and p be the number of pairs of vertices $v_i, v_j \in V$ such that both $(v_i, v_j), (v_j, v_i) \in A$. Hence, $p = \frac{|A-K|}{2}$.

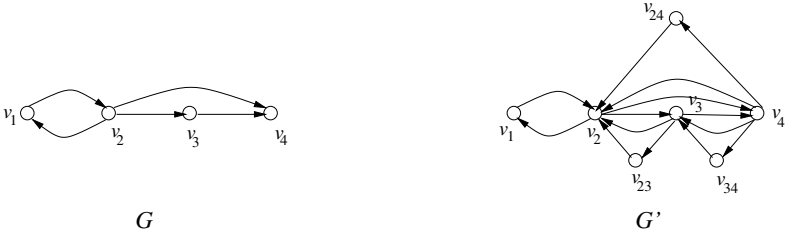


Fig. 1. A digraph G and the corresponding digraph G'

It is not difficult to establish the following claims.

Claim 1 Let (V', S') be a maximal SUBDAG of G' and $S = S' \cap A$. Then (V, S) is a SUBDAG of G and $|S'| = 3k + 2p - |S|$.

Claim 2 If (V', S'_o) is a minimum maximal SUBDAG of G', then (V, S_o) is a maximum SUBDAG of G. Also $|A| \leq 2|S_o|$.

Now $|S'_o| = 3k + 2p - |S_o| \leq 3(k + p) - |S_o| \leq 6|S_o| - |S_o| = 5|S_o|$. Also for any maximal SUBDAG (V', S') of G', $|S_o| - |S| = |S'| - |S'_o|$. □

Next we prove results about hardness of approximating MAX-MIN-VC and MAX-MIN-FVS, using reducibility arguments and the results of Håstad [12] concerning MAX-IS stated bellow.

Theorem 2. *[Håstad] Unless NP=ZPP (respectively P=NP), for any $\epsilon > 0$ there exists no polynomial time algorithm to approximate MAX-IS within a factor of $n^{1-\epsilon}$ (respectively $n^{\frac{1}{2}-\epsilon}$), where n is the number of vertices in an instance.*

Regarding MAX-MIN-VC we have

Theorem 3. *Unless NP = ZPP (respectively P=NP), for any $\epsilon > 0$ there exists no polynomial time algorithm to approximate MAX-MIN-VC within a factor of $\frac{1}{2}n^{\frac{1}{2}-\epsilon}$ (respectively $\frac{1}{2}n^{\frac{1}{4}-\epsilon}$), where n is the number of vertices in an instance.*

Proof. (Outline) Given an instance $G = (V, E)$ of MAX-IS, we construct an instance $G' = (V', E')$ of MAX-MIN-VC, where $V' = V \cup [\cup_{v \in V} \{v^1, v^2, \ldots v^{n+1}\}]$ and $E' = E \cup \{\{v, v^1\}, \{v, v^2\}, \ldots \{v, v^{n+1}\} | v \in V\}$. In other words, G' is obtained from G by introducing for each vertex $v \in V$, $n + 1$ additional vertices $v^1, v^2, \ldots, v^{n+1}$ and adding $(n+1)$ additional edges $\{v, v^1\}, \{v, v^2\}, \ldots \{v, v^{n+1}\}$ to the graph G.

We can establish the following claims without much difficulty.

Claim 1 A vertex cover $S' \subseteq V'$ of G' is a minimal VC iff (a) for $v \in S' \cap V$, $v^i \notin S'$, for any $1 \leq i \leq n + 1$, and (b) for $v \in V - S'$, $\{v^1, v^2, \ldots v^{n+1}\} \subseteq S'$.

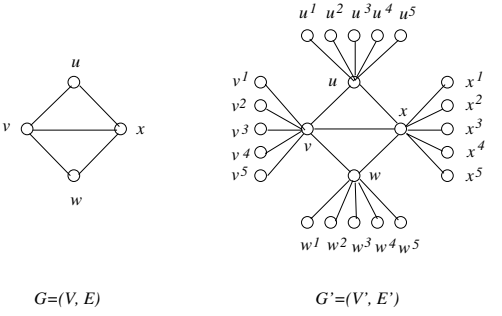


Fig. 2. An instance G of MAX-IS and the corresponding instance G' of MIN-MAX-VC

From the Claim 1 it follows that for any minimal VC $S' \subseteq V'$ of G', there exists a set $S \subseteq V$ such that $S' = (V - S) \cup [\cup_{v \in S}\{v^1, v^2, \ldots, v^{n+1}\}]$.

Claim 2 Let S be a maximal IS of G. Then $S' = (V-S)\cup[\cup_{v\in S}\{v^1, v^2, \ldots, v^{n+1}\}]$ is a minimal VC of G'.

Claim 3 Let S' be a minimal VC in G'. If $V - S'$ is not a maximal IS of G, then there exists a minimal VC S'' of G' such that $V - S''$ is a maximal IS of G and moreover,
(a) $|S''| > |S'|$
(b) $|V - S''| > |V - S'|$ and
(c) $|S''| = n(|V - S''| + 1)$.
Proof. Note that, for any VC S' of G', $V \cap S'$ is a VC of G. Hence $V - S' = V - (V \cap S')$ is an independent set of G. Let S' be a minimal VC of G' for which $V - S'$ is not a maximal independent set of G. Then we can always extend $(V - S')$ to a unique maximal IS S of G (in polynomial time) by introducing vertices of G one by one in the order $v_1, v_2, \ldots, v_n$ while mentaining the independence property. Hence $S \supset (V - S')$. By Claim 2, $S'' = (V - S) \cup [\cup_{v\in S}\{v^1, v^2, \ldots, v^{n+1}\}]$ is a minimal VC of G' and $|S''| = n(|S| + 1)$. Now we show that $S = V - S''$. For this first note that $S \subseteq V$ as S is a maximal independent set of G. Next, let $u \in S$, then from the definition of S'' it follows that $u \notin S''$, so $u \in V - S''$. Hence $S \subseteq V - S''$. Also, if $u \in V - S''$, then $u \notin S''$, i.e. $u \notin V - S$, so $u \in S$. Hence $S \supseteq V - S''$. Thus $S = V - S''$. From this it follows that $V - S''$ is a maximal independent set of G.

From Claim 1, we have $|S'| = |V \cap S'| + (n + 1)|V - (V \cap S')| = n(n + 1) - n|V \cap S'| = n + n|V - S'| = n(|V - S'| + 1)$. Since $|S| > |V - S''|$, it follows that $|S''| > |S'|$. Also (b) and (c) follow from the fact that $S = V - S''$. □

Claim 4 $S \subseteq V$ is a maximum IS of G iff $S' = (V-S)\cup[\cup_{v\in S}\{v^1, v^2, \ldots, v^{n+1}\}]$ is a maximum minimal VC of G'.
Proof. Let S be a maximum IS of G. By Claim 2, S' is a minimal VC of G'. If S' is not a maximum minimal VC of G', then using Claim 3 there exists a minimal VC S'' of G' such that $|S''| > |S'|$, $S = V - S''$ is a maximal IS of G and $|S''| = n(|S| + 1)$. As $|S'| < |S''|$, $|S'| = n + n|S|$ and $|S''| = n + n|\overline{S}|$, it follows

that $|S| < |\overline{S}|$, which is a contradiction. Hence S' is a maximum cardinality minimal VC in G.

Let S' be a maximum minimal VC of G'. Then by Claim 3, $S = V - S'$ is a maximal IS of G and $|S'| = n(|S| + 1)$. We claim that S is a maximum IS in G. Suppose there exists a maximal IS $S^* \subseteq V$ of G with $|S^*| > |S|$. By Claim 2, $\hat{S} = (V - S^*) \cup [\cup_{v \in S^*}\{v^1, v^2, \ldots, v^{n+1}\}]$ is a minimal VC in G' and $|\hat{S}| = n(|S^*| + 1)$. Since $|S^*| > |S|$, it follows that $|\hat{S}| > |S'|$, which is a contradiction. Hence, S is a maximum IS of G. □

Let $\alpha(G)$ denote the independence number and $\beta(G)$ denote the size of a maximum minimal VC in G. Hence, from Claim 4, we have $\beta(G') = n(\alpha(G)+1)$. Now let S' be any minimal VC of G'. If $V - S'$ is a maximal IS of G then to S' we associate $S = V - S'$ as the feasible solution of MAX-IS for G. If $V - S'$ is not a maximal IS of G then let S'' be the minimal VC of G' corresponding to S' as in Claim 3, so that $S = V - S''$ is a maximal IS of G and $|S'| < |S''| = n(|S| + 1)$. To this minimal VC S' of G' we associate S as the feasible solution of MAX-IS for G. Hence for any minimal VC S' of G' we have

$$\begin{aligned}
\frac{\alpha(G)}{|S|} &= \frac{n\alpha(G)}{n|S|} = \frac{\beta(G') - n}{|S''| - n} = \frac{\beta(G')}{|S''| - n} - \frac{n}{|S''| - n} \\
&= \frac{\beta(G')}{|S''|} \cdot \frac{|S''|}{|S''| - n} - \frac{1}{|S|} = \frac{\beta(G')}{|S''|} \cdot \frac{n(|S| + n)}{n|S|} - \frac{1}{|S|} \\
&= \frac{\beta(G')}{|S''|} + \frac{1}{|S|}\left(\frac{\beta(G')}{|S''|} - 1\right) \\
&\leq \frac{\beta(G')}{|S''|} + \frac{\beta(G')}{|S''|} - 1 \quad \left(\text{since } \frac{\beta(G')}{|S''|} \geq 1 \text{ and } |S| \geq 1\right) \\
&< 2\frac{\beta(G')}{|S''|} \leq 2\frac{\beta(G')}{|S'|}
\end{aligned}$$

Let N be the number of vertices in G'. Since $N = n^2 + 2n$ and $N \leq 2n^2$, for $n > 2$. Now, for any $\epsilon > 0$, $n^{1-\epsilon} \geq \frac{N^{\frac{1}{2}(1-\epsilon)}}{2^{\frac{1}{2}(1-\epsilon)}} \geq \frac{1}{2}N^{\frac{1}{2}(1-\epsilon)} \cdot 2^{\frac{1}{2}+\frac{\epsilon}{2}} \geq \frac{1}{2}N^{\frac{1}{2}(1-\epsilon)}$, and $n^{\frac{1}{2}-\epsilon} \geq \frac{1}{2}N^{\frac{1}{4}(1-2\epsilon)}$. Hence by, Theorem 2, the result follows. □

Regarding MAX-MIN-FVS, we have similar results.

Theorem 4. *Unless NP=ZPP (respectively P=NP), for any $\epsilon > 0$, there exists no polynomial time algorithm to approximate MAX-MIN-FVS within a factor of $\frac{1}{4}n^{\frac{1}{2}-\epsilon}$ (respectively $\frac{1}{4}n^{\frac{1}{4}-\epsilon}$), where n is the number of vertices in an instance.*

Proof. (Outline) We prove this by a reduction from MAX-MIN-VC to MAX-MIN-FVS as follows.

Let $G = (V, E)$ be a graph (an instance of MAX-MIN-VC). Construct an instance $G' = (V', A')$ of MAX-MIN-FVS from G with $V' = \cup_{v_i \in V}\{v_i^1, v_i^2\}$ and $A' = [\cup_{v_i \in V}\{(v_i^1, v_i^2)\}] \cup [\cup_{\{v_i, v_j\} \in E}\{(v_i^2, v_j^1), (v_j^2, v_i^1)\}]$. In other words, for each $v_i \in V$, G' has 2 vertices v_i^1, v_i^2 and an arcs (v_i^1, v_i^2). Also for each $\{v_i, v_j\} \in E$ G' has (v_i^2, v_j^1) and (v_j^2, v_i^1). Hence, G' has $2n$ vertices and $n + 2m$ arcs.

We can easily establish the following claims.

Claim 1 For any $C \subseteq V$,
(1) C is a VC of G iff $F = \{v_i^1 | v_i \in C\}$ is an FVS of G'.
(2) C is a minimal VC of G iff F is a minimal FVS of G'.
Claim 2 Let F be any minimal FVS of G'. Then
(1) for any $v_i \in V$, $F \cap \{v_i^1, v_i^2\}$ is either empty or singleton.
(2) for any $v_i \in V$ such that $F \cap \{v_i^1, v_i^2\} \neq \phi$, $F' = F - \{v_i^1, v_i^2\} + v_i^1$ is also a minimal FVS of G'.
(3) There is a minimal FVS F' of G' such that $|F'| = |F|$ and $F' = \{v_i^1 | v_i \in C\}$ for some minimal VC C of G such that $|C| = |F'|$.

Now let F_o be a maximum minimal FVS of G' and F be any minimal FVS of G'. By Claim 2, without loss of generality we can assume that every vertex in F_o (respectively, in F) is v_i^1 for some $v_i \in V$. Also by Claim 2, $C_o = \{v_i | v_i^1 \in F_o\}$, (respectively, $C = \{v_i | v_i^1 \in F\}$) is a maximum minimal VC (respectively, mininal VC) of G, and $|C_o| = |F_o|$ (respectively, $|C| = |F|$). Hence $\frac{|C_o|}{|C|} = \frac{|F_o|}{|F|}$.

Let $N = |V'|$. Then $N = 2n$. Now $\frac{1}{2}n^{\frac{1}{2}-\epsilon} = \frac{1}{2}\frac{(2n)^{\frac{1}{2}-\epsilon}}{2^{\frac{1}{2}-\epsilon}} = \frac{1}{4}N^{\frac{1}{2}-\epsilon} \cdot 2^{\frac{1}{2}+\epsilon} \geq \frac{1}{4}N^{\frac{1}{2}-\epsilon}$. Hence by, Theorem 3, the result follows. □

4 Hardness Results for Bounded Degree Digraphs

We know that MIN-FAS is APX-hard [14] and MAX-SUBDAG is APX-complete [17] for general digraphs. In this section, we show that these problems remain APX-hard even for k-total-regular digraphs for all $k \geq 4$. We also show that MIN-MAX-SUBDAG (respectively, MAX-MIN-VC) is APX-hard for digraphs of maximum total degree 12 (respectively, graphs of maximum degree 5). Regarding MIN-FAS, we first prove the following.

Lemma 1. *MIN-FAS-k $\leq_L$ MIN-FAS-$(k+1)$, for all $k \geq 1$.*

Proof. We construct in polynomial time, from a k-total-regular digraph $G = (V, A)$, a $(k+1)$-total-regular digraph $G' = (V', A')$ where $V' = V^1 \cup V^2$ where $V^i = \{v^i | v \in V\}$ for $i = 1, 2$ and $A' = A^1 \cup A^2 \cup B$ where $A^i = \{(u^i, v^i) | (u, v) \in A\}$ for $i = 1, 2$ and $B = \{(v^1, v^2) | v \in V\}$. From a minimal FAS S' of G' construct a minimal FAS S of G as follows: $S = \{(u, v) | (u^1, v^1) \in S^1\}$ where without loss of generality we assume that $S' = S^1 \cup S^2$ with S^1 and S^2 are minimal FASs of $G^1 = (V^1, A^1)$ and $G^2 = (V^2, A^2)$ respectively and $|S^1| \leq |S^2|$. It is easy to see that, if S'_o is a minimum FAS of G', then the corresponding S_o is a minimum FAS of G and $|S'_o| = 2|S_o|$. Further, for any minimal FAS $S' = S^1 \cup S^2$ of G', with $|S^1| \leq |S^2|$, $|S'| - |S'_o| = |S'| + |S_o| - 2|S_o| \geq 2(|S'| - |S_o|)$ so that $|S| - |S_o| \leq \frac{1}{2}(|S'| - |S'_o|)$. Thus, the two inequalities of L-reduction hold with $\alpha = 1$ and $\beta = \frac{1}{2}$. □

We now have the following.

Theorem 5. *MIN-FAS-k is APX-hard for all $k \geq 4$.*

Proof. (Outline) By Lemma 1, it is enough to show that MIN-FAS-4 is APX-hard. For this we show that MIN-VC-3 $\leq_L$ MIN-FAS-4.

We construct in polynomial time, from any 3-regular graph $G = (V, E)$ a 4-total-regular digraph $G' = (V', A')$ as defined in the proof of Theorem 4. For any FAS F of G', we associate a VC C of G defined as $C = \{v |$ either $(u^2, v^1) \in F$ or $(v^1, v^2) \in F\}$.

Further, C is a VC of G with $|C| \leq |F|$. For every edge $\{u, v\} \in E$, as $(u^1, u^2, v^1, v^2, u^1)$ is a cycle in G', F must contain at least one arc from this cycle, and so, C must contain either u or v. Hence, C is a VC of G, and by the construction of C from F, $|C| \leq |F|$.

Also, it can be easily shown that if F_o is a minimum FAS of G', then the associated VC C_o of G is a minimum VC of G and $|F_o| = |C_o|$, and for any FAS F of G', $|C| - |C_o| \leq |F| - |F_o|$. So the transformation from G to G' is an L-reduction with $\alpha = 1$ and $\beta = 1$. □

Similarly, for MAX-SUBDAG, we first prove the following.

Lemma 2. *MAX-SUBDAG-k* $\leq_L$ *MAX-SUBDAG-(k+1).*

Proof. Similar to the proof of Lemma 1. □

We now prove the following.

Theorem 6. *MAX-SUBDAG-k is APX-complete for any* $k \geq 4$.

Proof. By Lemma 2, it is enough to show that MAX-SUBDAG-4 is APX-hard. For this we show that MIN-VC-3 $\leq_L$ MAX-SUBDAG-4 and the reduction given in the proof of Theorem 5 is in fact an L-reduction from MIN-VC-3 to MAX-SUBDAG-4 with $\alpha = 1$ and $\beta = 1$. □

Regarding MIN-MAX-SUBDAG, we have the following easy theorem.

Theorem 7. *MIN-MAX-SUBDAG*$\leq$*12 is APX-hard.*

Proof. In the proof of Theorem 1, we constructed an instance G' of MIN-MAX-SUBDAG from an instance G of MAX-SUBDAG in such a way that if G is 4-regular then, every vertex in G' is of total degree at most 12. Since MAX-SUBDAG-4 is APX-complete, the result follows. □

Next we shall consider MAX-MIN-VC. First we have the following two simple lemmas.

Lemma 3. *For any 3-regular graph* $G = (V, E)$ *and any maximal IS* I *in* G, $|I| \geq \frac{1}{4}|V|$.

Lemma 4. *MAX-MIN-VC is k-approximable for graphs of maximum degree k,* $k \geq 1$*, and having no isolated vertex.*

Proof. Any minimal VC for such a graph is k-approximable. □

Now we have

Theorem 8. *MAX-MIN-VC*$\leq$*5 is APX-complete.*

Proof. Since MAX-MIN-VC is in class APX for bounded degree graphs (Lemma 4) and MAX-IS-3 is APX-complete [1], it is enough to show that MAX-IS-3 $\leq_L$ MAX-MIN-VC$\leq$5.

Let $G = (V, E)$ be a 3-regular graph. From G construct $G' = (V', E')$ of degree at most 5 as follows: $V' = V \cup [\cup_{v \in V} \{v^1, v^2\}]$ and $E' = E \cup [\cup_{v \in v} \{\{v, v^1\}, \{v, v^2\}\}]$.

By using the arguments given in the proof of Theorem 3, it can be proved that any minimal VC C of G' is of the form $C = (V - I) \cup [\cup_{v \in I} \{v^1, v^2\}]$, for some IS I of G where $I = V - (C \cap V)$ and $|C| = |I| + n$. Also, C_o is a maximum minimal VC of G' iff the associated I_o is a maximum IS of G, with $|C_o| = |I_o| + n$.

Now, $|C_o| = |I_o| + n \leq |I_o| + 4|I_o| = 5|I_o|$ (by Lemma 3), so that, the first inequality of L-reduction holds with $\alpha = 5$. Next, for any minimal VC C of G', $|C_o| - |C| = |I_o| + n - |I| - n = |I_o| - |I|$, so that, the second inequality of L-reduction holds with $\beta = 1$. □

Theorem 9. *MAX-MIN-FVS$\leq$10 is APX-hard.*

Proof. In the proof of Theorem 4, we constructed an instance G' of MAX-MIN-FVS from an instance G of MAX-MIN-VC in such a way that if G is of degree at most 5, then G' is of total-degree at most 10. Since MAX-MIN-VC$\leq$ 5 is APX-complete it follows that MAX-MIN-FVS$\leq$10 is APX-hard. □

5 Hardness Results for Bounded Degree Graphs

In this section we establish APX-hardness of MIN-FVS and MAX-MIN-FVS for certain restricted class of undirected graphs. Regarding MIN-FVS, it is known that it can be solved in polynomial time for all graphs of maximum degree 3 [22], but it is not known whether MIN-FVS is NP-complete for graphs of maximum degree 4 or 5. However, it is easy to show that [8] MIN-W-FVS$\leq$4 is NP-complete and also APX-complete.

Next we show that MIN-FVS-6 is APX-complete.

Theorem 10. *MIN-FVS-6 is APX-complete.*

Proof. (Outline) As MIN-FVS is in class APX [3], it is enough to show that MIN-FVS-6 is APX-hard. Towards this we will show that MIN-VC-3 $\leq_L$ MIN-FVS-6.

Let $G = (V, E)$ be a 3-regular graph. From G construct a 6-regular graph $G' = (V', E')$ as follows: For every edge $\{v_i, v_j\} \in E$, let $V_{ij} = \{v_{ij}^1, v_{ij}^2, v_{ij}^3, v_{ij}^4, v_{ij}^5, v_{ij}^6, v_{ij}^7\}$ be the set of seven new vertices and $H_{ij} = (V_{ij}, E_{ij})$ be the graph obtained from the complete graph on V_{ij} by removing the edge $\{v_{ij}^1, v_{ij}^7\}$. Now $V' = V \cup [\cup_{\{v_i, v_j\} \in E} V_{ij}]$ and $E' = E \cup_{\{v_i, v_j\} \in E} [E_{ij} \cup \{\{v_i, v_{ij}^1\}, \{v_{ij}^7, v_j\}\}]$, see Figure 3. Clearly G' is 6-regular.

Let F be an FVS of G'. Then F contains at least 4 vertices from V_{ij}. The following claims can be easily established.

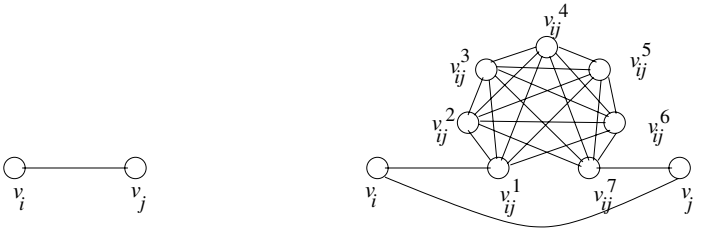


Fig. 3. An edge $\{v_i, v_j\} \in E$ and corresponding subgraph in G'.

Claim 1 Let F be any FVS of G' containing exactly 4 vertices from V_{ij} for some $\{v_i, v_j\} \in E$. Then F must contain either v_i or v_j.

To an FVS F of G', we associate the set C of vertices in G defined as $C = (F \cap V) \cup \{v_i \mid |F \cap V_{ij}| \geq 5 \text{ and } i < j\}$.

Claim 2 C is a VC of G and $|F| \geq |C| + 4|E| = |C| + 6n$.

Proof. If C is not a VC of G, then there exists $\{v_i, v_j\} \in E$ such that $C \cap \{v_i, v_j\} = \phi$. By the definition of C, it follows that $|F \cap V_{ij}| \leq 4$ and $F \cap \{v_i, v_j\} = \phi$. If $|F \cap V_{ij}| < 4$, then F is not an FVS of G', so $|F \cap V_{ij}| = 4$. By Claim 1, F must contain either v_i or v_j. Otherwise F can not be an FVS of G'. This contradicts that $F \cap \{v_i, v_j\} = \phi$. Hence, C is a VC of G.

Now $|F| = 4|E| + |F \cap V| + |\{v_i \mid |F \cap V_{ij}| \geq 5, i < j\}|$, as F contains at least 4 vertices from V_{ij} for each $\{v_i, v_j\} \in E$, and for the edges $\{v_i, v_j\} \in E$ such that $|F \cap V_{ij}| \geq 5$, F contains at least one more vertex from V_{ij} in addition to 4 vertices already considered. Hence, $|F| \geq |C| + 4|E| = |C| + 6n$ as G is a 3-regular and $|E| = \frac{3}{2}n$. □

Claim 3 For any VC C in G, the set $F = C \cup \{v_{ij}^2, v_{ij}^3, v_{ij}^4, v_{ij}^5\} | \{v_i, v_j\} \in E\}$ is an FVS of G' such that $C = F \cap V$ and $|F| = |C| + 6n$.

Claim 4 If F_o is a minimum FVS of G', then the associated set C_o is a minimum VC of G and $|F_o| = |C_o| + n$.

Now, $|F_o| = |C_o| + 6n \leq |C_o| + 12|C_o| = 13|C_o|$ (as any VC in a 3-regular graph contains at least $\frac{n}{2}$ vertices). Hence, the first inequality of L-reduction holds with $\alpha = 13$. Next, for any FVS F of G', $|F| - |F_o| \geq |C| + 6n - |C_o| - 6n = |C| - |C_o|$. So the second inequality of L-reduction holds with $\beta = 1$. □

Next we shall consider MAX-MIN-FVS. Before that we note the following.

Lemma 5. *For any FVS F of a 6-regular graph $G = (V, E)$, $|F| > \frac{2}{5}n$.*

Finally, we have,

Theorem 11. *MAX-MIN-FVS$\leq$9 is APX-hard.*

Proof. Let $G = (V, E)$ be a 6-regular graph. Construct a graph $G' = (V', E')$ of degree at most 9 as follows: $V' = V \cup \{v^1, v^2, v^3 | v \in V\}$ and $E' = E \cup \{(v, v^1), (v, v^2), (v, v^3), (v^1, v^2), (v^1, v^3) \mid v \in V\}$ (see Figure 4). Let F be any minimal FVS of G'. Note that, for any $v \in V - F$, F contains either v^1 or both v^2 and v^3. Further, if $v \in F \cap V$, then $F \cap \{v^1, v^2, v^3\} = \phi$. To F we associate

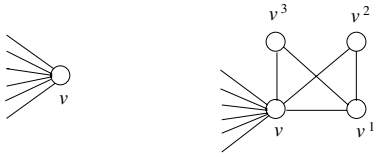


Fig. 4. a vertex v in G and its corresponding neighbors in G'

$C = F \cap V$, which is clearly an FVS of G. Note that $|F| \leq |C| + 2|V - F| = |C| + 2|V - C| = 2n - |C|$.

Let F_o be a maximum minimal FVS of G'. Then $|F_o| = 2n - |C_o|$ where $C_o = F_o \cap V$. For, if $|F_o| < 2n - |C_o|$, then $F = C_o \cup \{\{v^2, v^3\} | v \in V - C_o\}$ is a minimal FVS of G' with $|F| = 2n - |C_o| > |F_o|$ contradicting our assumption that F_o is a maximum minimal FVS of G'. Also note that C_o is a minimum FVS of G.

Now, $|F_o| = 2n - |C_o| < 5|C_o| - |C_o| = 4|C_o|$, (by previous Lemma). So, the first inequality of L-reduction holds with $\alpha = 4$. Next, for any minimal FVS F of G', $|F_o| - |F| \geq 2n - |C_o| - 2n + |C| = |C| - |C_o|$. So the second inequality of L-reduction holds with $\beta = 1$. □

6 Concluding Remarks

In this paper we have established hardness results for several NP-optimization problems related to MINLOP. These problems are variations or generalizations of well-known NP-optimization problems on graphs/digraphs. While for MAX-MIN-VC and MAX-MIN-FVS we have established strong results like those of Håstad [12] concerning MAX-IS and MAX-CLIQUE, for others we have just shown them to be APX-hard. Whether strong results about hardness of approximating such problems can be obtained is worth investigating. Despite such negative results, efforts may be made to obtain useful positive results giving efficient algorithms which may be $f(n)$-approximate for suitable function $f(n)$. Also, we do not have any results about MAX-MIN-FAS problem similar to MAX-MIN-FVS. These and other relavent issues concerning these problems are being pursued.

Acknowledgment: The authors thank C. R. Subramanian for a careful reading of an earlier draft and the anonymous referees for their comments and criticisms.

References

1. P. Alimonti AND V. Kann. Hardness of approximating problems on cubic graphs. in *Proc. 3rd Italian Conf. on Algorithms and Complexity*, LNCS-1203, Springer-Verlag (1997) 288-298.
2. G. Ausiello, P. Crescenzi AND M. Protasi. Fundamental Study: Approximate solution of NP optimization problems, *Theoretical Computer Science* 150 (1995) 1-55.

3. V. Bafna, P. Berman AND T. Fujito. Constant ratio approximations of feedback vertex sets in weighted undirected graphs, in *6th Annual International Symposium on Algorithms and Computation* (1995).
4. A. Chaudhary AND S. Vishwanathan. Approximation algorithms for achromatic number, *Proc. 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, ACM-SIAM, (1997) 558-563.
5. G. A. Cheston, G. Fricke, S. T. Hedetniemi AND D. P. Jacobs. On the computational complexity of upper fractional domination. *Discrete Appl. Math.*, 27 (1990) 195-207.
6. P. Crescenzi AND A. Panconesi. Completeness in approximation classes. *Information and Computation*, 93 (1991) 241-262.
7. P. Crescenzi, V. Kann, R. Silvestri AND L. Trevisan. Structures in approximation classes, in *1st. Annu. Int. Conf. on Computing and Combinatorics*, LNCS-959, Springer-Verlag, (1995) 539-548.
8. T. Fujito. Personal communication, 1999.
9. M. Grötschel, M. Jünger AND G. Reinelt. On the acyclic subgraph polytope, *Math. Programming* 33 (1985) 28-42.
10. F. Harary. "Graph Theory", Addition-Wesley, Reading, MA, 1969.
11. F. Harary. Maximum versus minimum invariants for graphs, *Jr. of Graph Theory*, 7 (1983) 275-284.
12. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$, In *Proc. 37th IEEE Sympo. on Foundation of Comput. Sci.* (1996) 627-636.
13. M. M. Haldórsson. Approximating the minimum maximal independence number, Info. Proc. Letters, 46 (1993) 169-172.
14. V. Kann, On the Approximability of NP-complete Optimization Problems, Ph. D. thesis, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.
15. S. Khanna, R. Motwani, M. Sudan AND U. Vazirani. On syntactic versus computational views of approximability, in *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science* (1994) 819-836.
16. C. Lund and M. Yannakakis. On the hardness of approximating minimization problems, *J, ACM*, 41 (1994) 960-981.
17. C. H. Papadimitriou AND M. Yannakakis. Optimization, Approximation, and Complexity Classes. *J. Comput. System Sci.* 43 (1991) 425-440.
18. K. Peters, R. Laskar AND S. T. Hedetniemi. Maximinimal/Minimaximal connectivity in graphs. *Ars Combinatoria*, 21 (1986) 59-70.
19. D. F. Manlove. Minimaximal and maximinimal optimization problems: a partial order-based approach, Ph. D. Thesis, University of Glasgow (1998).
20. S. Mishra AND K. Sikdar. On approximate Solutions of Linear Ordering Problems, T. R. No. - 5/98,Stat-Math Unit, Indian Statistical Institute, Calcutta.
21. S. Mishra AND K. Sikdar. On the hardness of approximating some NP-optimization problems related to minimum linear ordering problem, T. R. No. - 7/99,Stat-Math Unit, Indian Statistical Institute, Calcutta.
22. S. Ueno, Y. Kajtani AND S. Gotoh. On the nonseparating independent set problem and feedback set problem for graphs with no vertex exceeding three, *Disc. Math.*, 72 (1988) 355-360.

Maximum Clique and Minimum Clique Partition in Visibility Graphs*

Stephan Eidenbenz and Christoph Stamm

Institute for Theoretical Computer Science, ETH Zürich, Switzerland
{eidenben, stamm}@inf.ethz.ch

Abstract In an alternative approach to "characterizing" the graph class of visibility graphs of simple polygons, we study the problem of finding a maximum clique in the visibility graph of a simple polygon with n vertices. We show that this problem is very hard, if the input polygons are allowed to contain holes: a gap-preserving reduction from the maximum clique problem on general graphs implies that no polynomial time algorithm can achieve an approximation ratio of $\frac{n^{1/8-\epsilon}}{4}$ for any $\epsilon > 0$, unless $NP = P$. To demonstrate that allowing holes in the input polygons makes a major difference, we propose an $O(n^3)$ algorithm for the maximum clique problem on visibility graphs for polygons without holes (other $O(n^3)$ algorithms for this problem are already known [3,6,7]). Our algorithm also finds the maximum weight clique, if the polygon vertices are weighted.
We then proceed to study the problem of partitioning the vertices of a visibility graph of a polygon into a minimum number of cliques. This problem is APX-hard for polygons without holes (i.e., there exists a constant $\gamma > 0$ such that no polynomial time algorithm can achieve an approximation ratio of $1 + \gamma$). We present an approximation algorithm for the problem that achieves a logarithmic approximation ratio by iteratively applying the algorithm for finding maximum weighted cliques. Finally, we show that the problem of partitioning the vertices of a visibility graph of a polygon *with* holes cannot be approximated with a ratio of $\frac{n^{1/14-\gamma}}{4}$ for any $\gamma > 0$ by proposing a gap-preserving reduction. Thus, the presence of holes in the input polygons makes this partitioning problem provably harder.

1 Introduction

Visibility problems have received considerable attention in the past. On the one hand, art gallery problems – such as Minimum Vertex Guard – have been studied intensively with respect to both, bounds on descriptional complexity as well as computational complexity results. On the other hand, visibility graphs continue to draw interest. A *simple polygon with(out) holes* is given by its ordered sequence of vertices on the outer boundary, together with an ordered sequence

* We gratefully acknowledge the support of this work by the Swiss National Science Foundation.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 200–212, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of vertices for each hole, if any. Two polygon vertices *see* each other, iff the straight line segment connecting the two vertices does not intersect the exterior (or holes) of the polygon. A graph $G = (V, E)$ with vertices $v_1, \ldots, v_n$ is a *visibility graph*, iff there exists a simple polygon P (with or without holes) consisting of vertices $p_1, \ldots, p_n$ such that the polygon vertices p_i and p_j see each other, iff $(v_i, v_j) \in E$. The *visibility graph characterization* problem consists of finding a set of graph-theoretic properties that exactly define visibility graphs. It is closely related to the *visibility graph recognition* problem, which consists of determining if a given graph is a visibility graph. A lot of work has been done on the visibility graph characterization problem (see [15,13,24] or [25] for a survey), but it still is not satisfactorily solved. A different approach to "characterizing" the class of visibility graphs is to determine the computational complexity (and in case of NP-hardness the approximability) of classic graph-theoretic problems on visibility graphs. Actually, a considerable amount of work has been done that falls in the realm of this approach, because many classic graph-theoretic problems have a geometric interpretation in the context of visibility graphs. Also, the problem Minimum Coloring on Visibility Graph is mentioned as an open problem (with respect to its computational complexity) in an open problems list [22].

Consider, for example, the problem Maximum Independent Set on Visibility Graph, in which we are given a simple polygon with n vertices and we are to find the maximum independent set in the corresponding visibility graph. This problem corresponds to finding a maximum set of polygon vertices that are hidden from each other. The problem is therefore also called Maximum Hidden Vertex Set. It is known to be NP-hard [26], APX-hard for polygons without holes and hard to approximate with an approximation ratio of $\frac{n^{\frac{1}{6}-\gamma}}{4}$ for all $\gamma > 0$ for polygons with holes [12].

The problem Minimum Dominating Set on Visibility Graph corresponds to finding a minimum set C of polygon vertices such that each polygon vertex can be seen from at least one vertex in C. This problem is a variation of the well known art gallery problem Minimum Vertex Guard, which asks for a minimum number of vertices (guards) of a given polygon such that every point in the interior and on the boundary of the polygon can be seen from at least one guard. It is easy to see that the inapproximability results as well as approximability results for Minimum Vertex Guard carry over to Minimum Dominating Set on Visibility Graph, which therefore is APX-hard [11] for polygons without holes and not approximable with some approximation ratio that is logarithmic in the number of polygon vertices for polygons with holes [9]. Furthermore it is approximable with a logarithmic ratio [14].

In this paper we study the problem Maximum Clique on Visibility Graph with(out) Holes, in which we are given a simple polygon with(out) holes with n vertices and we are to find the largest clique in the corresponding visibility graph. We distinguish two separate problems by allowing holes or not. Note that in the case of polygons without holes, this problem corresponds to finding a largest (with respect to number of vertices) convex subpolygon of a given polygon. The geometric interpretation in the case of polygons with holes is unclear.

This problem has potential applications in the setting up of antenna networks in terrains (see [10,12] for the relationship of polygons with terrains), where all antennas must see each other in order to guarantee optimum connectivity.

We show that Maximum Clique on Visibility Graph with Holes cannot be approximated by any polynomial time algorithm with an approximation ratio of $\frac{n^{1/8-\epsilon}}{4}$ for any $\epsilon > 0$, unless $NP = P$ in Sect. 2. Thus, Maximum Clique on Visibility Graph with Holes is almost as hard to approximate as clique on general graphs. We propose a gap-preserving reduction (a technique introduced in [2]) from Maximum Clique on general graphs to get this result.

The problem Maximum Clique on Visibility Graph without Holes is known to be solvable in time $O(n^3)$ by slightly adopting algorithms [3,6,7] that were developed to solve different problems (such as finding empty convex polygons that are maximum with respect to the number of vertices by connecting some of the input points). We propose an additional $O(n^3)$ algorithm for this problem for polygons without holes in Sect. 3, which uses dynamic programming. Our method also solves the weighted version of this problem, in which each vertex is assigned a weight value and the total weight of all vertices in the clique is to be maximized. We will use this weighted version (only with weights 0 and 1) to obtain an approximation algorithm for another problem (see Sect. 4).[1]

This gap of "solvable in cubic time" vs. "almost as hard to approximate as clique" is the most extreme gap ever discovered between the two versions of a visibility problem on polygons with vs. without holes.

The problem Minimum Clique Partition consists of finding a partitioning of the vertices of a given graph into a minimum number of disjoint vertex sets, each of which must be a clique in the graph. Again, we can define this problem on visibility graphs of polygons with or without holes. In the case of polygon without holes, this problem is closely related to the problem Minimum Convex Cover without Holes, which consists of covering a given polygon without holes with a minimum number of (possibly overlapping) convex polygons. Minimum Clique Partition on Visibility Graphs without Holes is a variant of Minimum Convex Cover without Holes, where only the vertices are of interest (not the edges or the interior area of the polygon).

A careful analysis (presented in [8]) of the reduction that was originally constructed to show the NP-hardness of Minimum Convex Cover [5] reveals that Minimum Convex Cover is APX-hard. The analysis can be easily adopted to work for Minimum Clique Partition on Visibility Graphs without Holes. Therefore, Minimum Clique Partition on Visibility Graphs without Holes is APX-hard[2], i.e. there exists a constant $\epsilon > 0$ such that no polynomial time approximation algorithm can achieve an approximation ratio of $1+\epsilon$ for these problems, unless $NP = P$. In Sect. 4, we propose an approximation

[1] The fact that our $O(n^3)$ algorithm solves the *weighted* version of Minimum Clique on Visibility Graph without Holes, which will be used as a major building block for another approximation algorithm, is the main reason for including it in this paper, next to the obvious reason of self-containment.

[2] See [4] and [2] for an introduction to the class APX.

algorithm for Minimum Clique Partition on Visibility Graphs without Holes that iteratively applies the algorithm for the weighted version of Maximum Clique on Visibility Graph without Holes and show that it achieves a logarithmic approximation ratio. This result sheds some light on the approximability of Minimum Clique Partition on Visibility Graphs without Holes, but it still is not known whether a constant approximation ratio can be achieved or whether the logarithmic approximation algorithm presented is optimum.

There seems to be no straightforward geometric interpretation of Maximum Clique Partition on Visibility Graph with Holes, but the problem is certainly of theoretic interest, as we propose a gap-preserving reduction in Sect. 5 from Maximum Clique Partition on general graphs that shows that Maximum Clique Partition on Visibility Graph with Holes cannot be approximated with an approximation ratio of $\frac{n^{1/14-\gamma}}{4}$ for any $\gamma > 0$.

This is the first result for a visibility problem that is NP-hard no matter whether holes are allowed or not, where we are able to show that the approximation properties are clearly different for the cases of polygons with vs. without holes: While Maximum Clique Partition on Visibility Graph with Holes cannot be approximated with an approximation ratio of $\frac{n^{1/14-\gamma}}{4}$ for any $\gamma > 0$, we have a logarithmic approximation algorithm for Minimum Clique Partition on Visibility Graphs without Holes.

In Sect. 6, we draw conclusions.

As for related work other than the previously mentioned, there are several surveys on art gallery and visibility problems [21] [25] [27]. As for computational complexity results, Minimum Convex Cover with(out) Holes can be approximated with a logarithmic approximation ratio [8]. The problems Minimum Vertex/Edge/Point Guard, which are guarding problems with different types of guards, are known to be NP-hard [19] and APX-hard [11] for polygons without holes, and inapproximable with an approximation ratio logarithmic in the number of polygon vertices for polygons with holes [9]. Furthermore, Minimum Vertex/Edge Guard can be approximated with a logarithmic approximation ratio for polygons with and without holes [14].

2 An Inapproximability Result for Maximum Clique on Visibility Graph with Holes

We propose a gap-preserving reduction from the Maximum Clique problem to the Maximum Clique on Visibility Graph with Holes problem. The technique of gap-preserving reductions [2] maps the promise problem of Maximum Clique to the promise problem Maximum Clique on Polygons with Holes. Suppose we are given an instance I of the promise problem Maximum Clique, i.e., a graph $G = (V, E)$ with $n := |V|$ and an integer k with $2 \leq k \leq n$, where $\epsilon > 0$ is arbitrarily small, but fixed. We are promised that the size of a maximum clique in the graph G is either at least k or strictly less than $\frac{k}{n^{1/2-\epsilon}}$. It

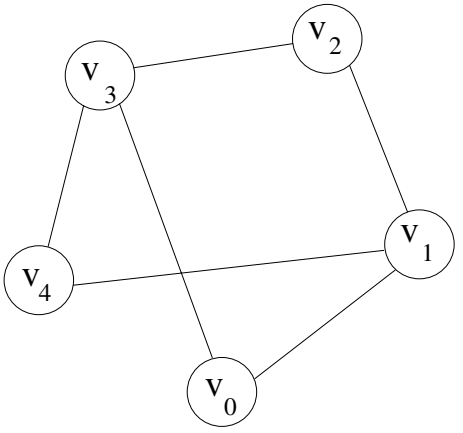


Figure 1. Basic construction: an input graph

is NP-hard to decide which of these two cases is true, because otherwise, Maximum Clique could be approximated by a polynomial time algorithm with an approximation ratio of $n^{1/2-\epsilon}$, which cannot be done unless $NP = P$ [16].

The basic idea of the reduction is shown in Figs. 1 and 2. For each instance I of Maximum Clique, i.e., for each graph $G = (V, E)$ with $n := |V|$ (as shown in an example in Fig. 1), we construct an instance I' of Maximum Clique on Visibility Graph With Holes, i.e., a polygon with holes (as shown in an example in Fig. 2). The main polygon is in the shape of a regular $2n$-gon with vertices named v_i and v'_i for $i \in \{1, \ldots, n\}$. For each vertex pair $(v_i, v_j) \notin E$, we construct two small triangular holes, one around the intersection point of the line segment from v_i to v_j and the line segment from v'_i to v'_{i+1}, and one around the intersection point of the line segment from v_i to v_j and the line segment from v'_j to v'_{j+1}. These triangular holes are designed to block the view of vertices v_i and v_j that are not supposed to see each other, since they are not connected by an edge in the input graph. The detailed, and rather technical construction of the holes is described in [12], and we therefore omit it here.

In order to make the reduction work, we refine the polygon with holes obtained thus far as follows:

For each vertex v_i let v_i^L (v_i^R) be the point on the line segment from v_i to v'_{i-1} (v'_i) that is closest to point v'_{i-1} (v'_i) such that the view of v_i^L to v_j^R (v_i^R to v_j^L) for all v_j is still blocked by the corresponding two holes, if vertices v_i and v_j are not connected in the input graph by an edge. These points are illustrated in Fig. 2.

For each vertex v_i, we replace the two line segments from v_i^L to v_i to v_i^R by a convex chain of of $n^3 - 1$ line segments (called the *chain of* v_i). This is illustrated

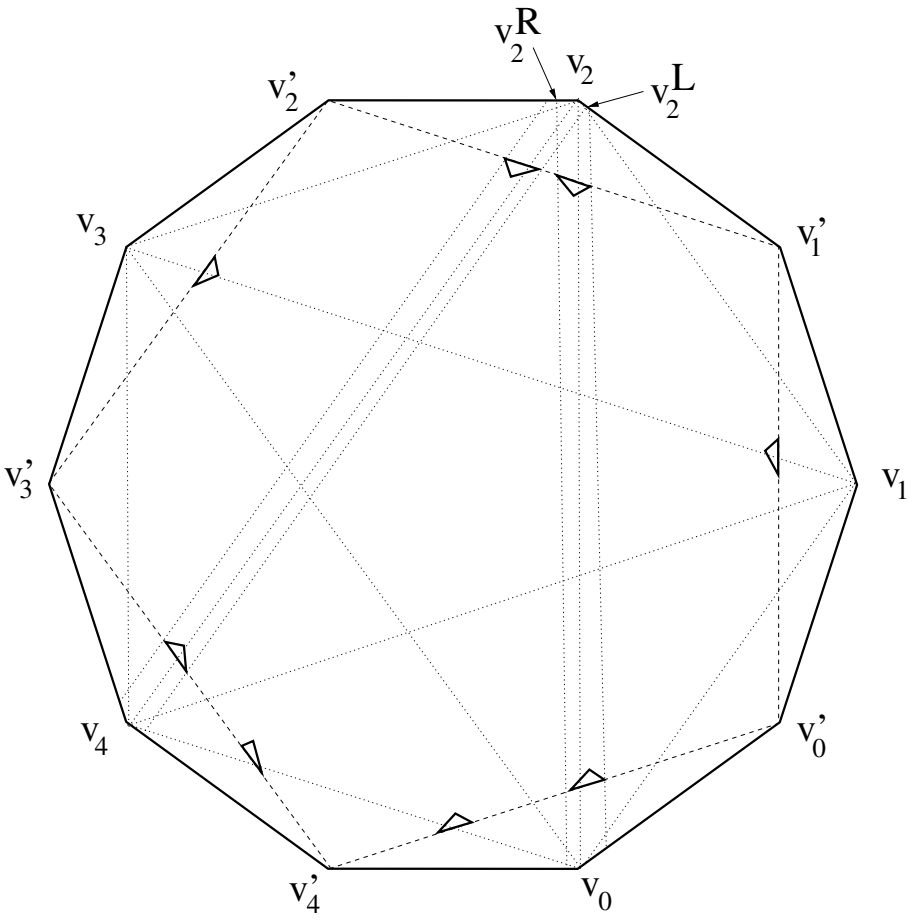


Figure 2. Basic construction: polygon with holes resulting from the graph in Fig. 1

in Fig. 3. By the way that we chose points v_i^L and v_i^R, it is ensured that any two vertices from chains of v_i and v_j see each other, iff $(v_i, v_j) \in E$.

The following two lemmas allow us to prove the main result of this section. Let OPT denote the size of an optimum solution of the MAXIMUM CLIQUE instance I and let OPT' denote the size of an optimum solution of the MAXIMUM CLIQUE ON VISIBILITY GRAPH WITH HOLES instance I'. Let $\epsilon > 0$.

Lemma 1. $OPT \geq k \implies OPT' \geq n^3 k$

Proof. If $OPT \geq k$, then there exists a clique of size k in I. We obtain a clique in I' of size $n^3 k$ by simply letting all the n^3 vertices of the chain of v_i be in the solution, if vertex $v_i \in V$ is in the clique. □

Lemma 2. $OPT < \frac{k}{n^{1/2-\epsilon}} \implies OPT' < \frac{n^3 k}{n^{1/2-\epsilon}} + 3n^2$

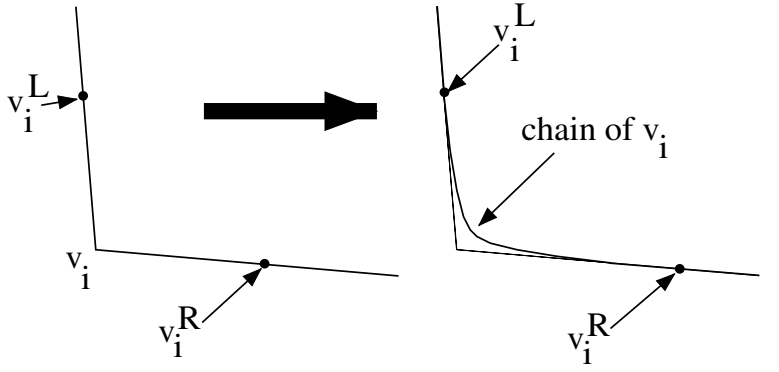


Figure 3. Chain of vertex v_i

Proof. We prove the contraposition:$OPT' \geq \frac{n^3k}{n^{1/2-\epsilon}} + 3n^2 \implies OPT \geq \frac{k}{n^{1/2-\epsilon}}$. Suppose we have a solution of I' with $\frac{n^3k}{n^{1/2-\epsilon}} + 3n^2$ points. Since there are at most $n(n-1)$ holes with 3 vertices each and n additional vertices v_i', there can be at most $3n(n-1)+n \leq 3n^2$ vertices in the clique that are not part of the chain of some v_i. Therefore, at least $\frac{n^3k}{n^{1/2-\epsilon}}$ vertices of the clique must be in chains. Since a chain consists of only n^3 vertices, each chain can contribute at most n^3 vertices to the clique. Therefore, the number of chains that contain at least one point from the solution is at least $\frac{\frac{n^3k}{n^{1/2-\epsilon}}}{n^3} = \frac{k}{n^{1/2-\epsilon}}$. Since no two vertices of two different chains v_i and v_j see each other unless $(v_i, v_j) \in E$, we immediately have a solution for I with at least $\frac{k}{n^{1-\epsilon}}$ vertices by letting v_i be in the clique if at least one point of the chain of v_i is in the solution. □

Lemmas 1 and 2 transform the promise problem of Maximum Clique as mentioned above into a promise problem of Maximum Clique on Visibility Graph with Holes, where we are promised that an optimum solution contains either at least n^3k vertices or strictly less than $\frac{n^3k}{n^{1/2-\epsilon}} + 3n^2$ vertices. It is also NP-hard to decide, which of the two cases is true, since otherwise, we could solve the NP-hard promise problem of Maximum Clique (see [2] for more details on the notion of such gap-preserving reductions). Maximum Clique on Visibility Graph with Holes can therefore not be approximated by any polynomial time approximation algorithm with an approximation ratio of:

$$\frac{n^3k}{\frac{n^3k}{n^{1/2-\epsilon}} + 3n^2} = \frac{n^3k}{\frac{n^3k+3n^{1-2\epsilon}}{n^{1/2-\epsilon}}} \geq \frac{n^3k}{\frac{2n^3k}{n^{1/2-\epsilon}}} = \frac{n^{1/2-\epsilon}}{2}$$

We now need to express the size $|I'|$ of the Maximum Clique on Visibility Graph with Holes instance I' by the size n of the Maximum Clique instance I. According to the construction, $|I'| \geq 2n^4$. We proceed:

$$\frac{n^{1/2-\epsilon}}{2} \geq \frac{\frac{|I'|^{\frac{1}{4}(\frac{1}{2}-\epsilon)}}{2^{\frac{1}{4}(\frac{1}{2}-\epsilon)}}}{2} \geq \frac{|I'|^{\frac{1}{8}-\frac{\epsilon}{4}}}{4}$$

This completes the proof of our main theorem of this section:

Theorem 1. Maximum Clique on Visibility Graphs with Holes *cannot be approximated by any polynomial time algorithm with an approximation ratio of* $\frac{|I'|^{1/8-\gamma}}{4}$*, where* $|I'|$ *is the number of vertices in the polygon and where* $\gamma > 0$*, unless* $NP = P$*.*

3 An $O(n^3)$ Algorithm for Maximum Clique on Visibility Graph without Holes

Our polynomial time algorithm for Maximum Clique on Visibility Graph without Holes uses dynamic programming.

Suppose we are given a simple polygon P without holes, which consists of n vertices $v_1, \ldots, v_n$ in counterclockwise order. We first compute the visibility graph $G = G(P)$ of this polygon, which can be done in time $O(|E|)$, where E is the set of edges in G [17]. This allows us to answer queries of the form "Does vertex v_i see vertex v_j ?" in time $O(1)$. As we will use a weighted version of this problem to find an approximation algorithm for Minimum Clique partition on Visibility Graph without Holes, we introduce a non-negative weight w_i for each vertex v_i. We are now to find a clique in G that has a maximum total weight. In the following, all operations are modulo n, where applicable. Let $A_{i,j,k}$ with $i < j \leq k$ be the *maximum* clique (with respect to its weight) among all cliques, which consist of vertices v_i, v_j and v_k and additional vertices $v_{j'}$ with $i < j' < j$. Let $|A_{i,j,k}|$ denote the weight of $A_{i,j,k}$. The optimum solution OPT is:

$$OPT = A_{i,j,j} \text{ where } i, j \text{ are such that } |A_{i,j,j}| = \max_{1 \leq i < j \leq n} |A_{i,j,j}|$$

Given all $A_{i,j,j}$, OPT can be computed in $O(n^2)$ time. A can be considered to be a three-dimensional table. It is initialized as follows:

$$A_{i,i+1,j} = \{v_i, v_{i+1}, v_j\}, \forall i, j, \text{ where vertices } v_i, v_{i+1}, v_j \text{ all see each other}$$

This initialization can be done in time $O(n^3)$. The remaining entries of the table A are initialized with empty sets and then computed according to Lemma 3.

Lemma 3. *Assume vertices* v_i*,* v_j*, and* v_k *see each other. Then,* $A_{i,j,k} = A_{i,j',j} \cup v_k$*, where* j' *is such that* $|A_{i,j',j}| = \max |A_{i,j'',j}|$*, where the maximum is taken over all* j'' *with* $i \leq j'' \leq j$ *and where* $v_{j''}$ *sees* v_i*,* v_j*, and* v_k*.*

Proof. The proof is inductive. Suppose we know that the lemma holds for $A_{i,j,k'}$ with $k' < k$. To show that it also holds for $A_{i,j,k}$, we assume by contradiction that there exists a clique P', which consists of vertices v_i, v_j and v_k and additional vertices $v_{l'}$ with $i < l' < j$ and which is strictly heavier than $A_{i,j,k}$ (as computed in the Lemma).

Let v_l be the vertex in P' that is the neighbor of v_j in P' in clockwise order, when we interpret the clique P' as a convex polygon. Now, consider the clique

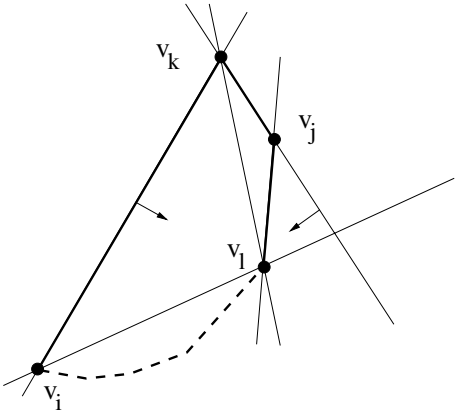


Figure 4. Proof of Lemma 3

$A_{i,l,j}$, which is maximum by assumption. Because v_j is the neighboring vertex of v_l in P', we have $|P'| \leq |A_{i,l,j}| + w_k$. We will now argue that vertex v_k can be added to the clique $A_{i,l,j}$ and the resulting set of vertices (i.e. $A_{i,l,j} \cup v_k$) is still a clique.

Consider Fig. 4. First, note that vertex v_j must lie to the right of the line from v_i to v_k, because vertices v_i, v_j and v_k all see each other and because $i \leq j \leq k$. Since $v_i, v_l, v_j, v_k \in P'$ and $i \leq l \leq j \leq k$ and since P' is a clique, vertex v_l must lie to the right of the line from vertex v_i to v_k and to the left of the line from v_j to v_k. Now, consider all vertices $l'' \in A_{i,l,j}$ that lie between i and l (i.e. $i < l'' < l$). By definition of $A_{i,l,j}$, all these vertices see v_i, v_l and v_j. This implies that all vertices $v_{l''}$ also see v_k, because any polygon segment blocking the view of some vertex $v_{l'}$ to v_k would imply the existence of a polygon segment that would block the view of $v_{l''}$ to either v_i or v_l. We have shown that all vertices in $A_{i,l,j}$ also see v_k, therefore $A_{i,l,j} \cup v_k$ is a clique as well.

The polygon $A_{i,l,k}$ is among those polygons over which the maximum is taken in the Lemma to compute $A_{i,j,k}$. Therefore, $|A_{i,j,k}| \geq P'$, which is a contradiction to the assumption that P' is strictly heavier than $A_{i,j,k}$. □

A trivial implementation of the algorithm thus suggested would have a running time of $O(n)$ for each of the $O(n^3)$ table entries, which results in an overall running time of $O(n^4)$. It is, however, possible to implement the algorithm with a total running time of $O(n^3)$. To achieve this, we show how to compute $A_{i,j,k}$ with i, j fixed and $A_{i,j',j}$ already computed for $i \leq j' \leq j$, in time $O(n)$ (for all k with $j \leq k \leq i$). This directly leads to an $O(n^3)$ algorithm, since there are only $O(n^2)$ pairs i, j.

To speed up the algorithm, fix i, j. Then compute all v_k with $j \leq k \leq i$ that are visible from v_i and v_j. Let K denote the counterclockwise ordered set of all these vertices v_k. Let L denote the clockwise ordered set of vertices v_l with $i \leq l \leq j$ that are visible to both v_i and v_j For each vertex $v_l \in L$ (working from

v_j towards v_i): Determine, which vertices $v_k \in K$ are visible from v_l. Let $k' < k''$. Note that if v_l sees $v_{k'} \in K$, then it also sees $v_{k''} \in K$. Let $v_{k_{\min}}$ denote the first $v_k \in K$ that sees v_l. It suffices to just "link" $v_{k_{\min}} \in K$ to $A_{i,l,j}$ (depending on the implementation, a "link" could be an entry in some record field or a pointer). Note that as we work our way through L from v_j to v_i, the $v_{k_{\min}}$'s get smaller, i.e. proceed towards v_j. Thus, determining $v_{k_{\min}}$ can be done in total time $O(|K|)$ for all $v_l \in L$ (if $(|K| > |L|$, otherwise it is $O(|L|)$). We now scan through K. If $v_k \in K$ is "linked" to some $A_{i,j,l}$, we compare the weight of $A_{i,j,l}$ with the weight of the currently optimum solution. If $|A_{i,j,l}|$ is greater than the weight of the currently optimum solution, we update the currently optimum solution to $A_{i,j,l}$. If v_k is not "linked", we link it to the currently optimum solution. Now, set $A_{i,j,k}$ to the currently optimum solution with v_k added. We also store $|A_{i,j,k}|$. This scanning through K can be done in time $O(|K|)$. Thus, the total running time to compute $A_{i,j,k}$ for all k is $O(\max\{|L|, |K|\})$, which is $O(n)$.

Let us summarize the result of this section:

Theorem 2. *The weighted version of* Maximum Clique on Visibility Graph without Holes*, where non-negative weights are assigned to the vertices, can be solved in time* $O(n^3)$ *using dynamic programming.*

4 An Approximation Algorithm for Minimum Clique Partition on Visibility Graph without Holes

Our approximation algorithm for Minimum Clique Partition on Visbility Graph without Holes iteratively applies the polynomial time algorithm for the weighted version of Maximum Clique on Visibility Graph without Holes. It works as follows for a given polygon P:

1. Compute the visibility graph $G(P)$ of the polygon P. Let all vertices have weight 1.
2. Find the maximum weighted clique C in $G(P)$ using the algorithm proposed in Sect. 3. Let all vertices $v_i \in C$ have weight 0. Add C to the solution S.
3. Repeat step 2 until there are no vertices with weight 1 left. Return S.

To obtain a performance guarantee of this algorithm, consider the Minimum Set Cover[3] instance I, which has all polygon vertices v_i as elements and the vertices of each clique in the visibility graph of the polygon are a set in I. The greedy heuristic for Minimum Set Cover, which consists of recursively adding to the solution a set, which contains a maximum number of elements not yet covered by the solution, achieves an approximation ratio of $1+\ln n$, where n is the number of elements in I [18]. Our algorithm works in exactly this way. Note that we do not have to compute all the sets of the Minimum Set Cover instance I (which would possibly be a number exponential in n), since it suffices to always

[3] Minimum Set Cover consists of finding a minimum number of sets among a given collection of sets such that each element of a given universe appears in at least one of these sets.

compute a set (or clique), which contains a maximum number of vertices not yet covered by the solution, which is achieved by reducing the weights of the vertices already in the solution to 0. Thus, our algorithm is polynomial.

Theorem 3. Minimum Clique Partition on Visibility Graph without Holes *can be approximated with an approximation ratio of* $1 + \ln n$*, where* n *is the number of polygon vertices, by a greedy heuristic.*

5 An Inapproximability Result for Minimum Clique Partition on Visibility Graph with Holes

Minimum Clique Partition on general graphs is equivalent to Minimum Graph Coloring [4]. It cannot be approximated by any polynomial time algorithm with an approximation ratio of $n^{1/7-\epsilon}$, where $\epsilon > 0$ and n is the number of vertices in the graph [4]. We propose a gap-preserving reduction from Minimum Clique Partition on general graphs to Minimum Clique Partition on Visibility Graph with Holes.

Again, we map the NP-hard promise problem of Minimum Clique Partition on general graphs, where we are promised that an optimum solution consists of either at most k or strictly more than $n^{1/7-\epsilon}k$ cliques, to a promise problem of Minimum Clique Partition on Visibility Graph with Holes, where we are promised that an optimum solution consists of either at most $k + 3$ or strictly more than $n^{1/7-\epsilon}k$ cliques. We use the same construction as used in Sect. 2. However, we do not need to use the "chains" as introduced in Sect. 2. Let OPT (OPT') denote the size of an optimum solution of the Maximum Clique Partition (Maximum Clique Partition on Visibility Graph with Holes) instance I (I'). Let $\epsilon > 0$.

Lemma 4. $OPT \leq k \implies OPT' \leq k + 3$ *and* $OPT > n^{1/7-\epsilon}k \implies OPT' > n^{1/7-\epsilon}k$

Proof. For the first implication: If $OPT \leq k$, then there exists a solution of size k in I. We obtain a solution in I' of size $k + 3$ by simply letting all cliques from the solution in I be cliques in I' and by adding three more cliques. One of these consists of all the "bottom" vertices of all holes (i.e. those vertices that lie on line segments between points v'_{i-1} and v'_i for all i). The holes are constructed in such a way that these vertices actually form a clique (see [12]). The second clique consists of the "top" vertices of all holes. The third clique consists of all vertices v'_i. The construction of the reduction ensures that these additional cliques actually are cliques.

We prove the contraposition of the second implication: A solution for I' can be interpreted as a solution for I, where the additional vertices of I' are ignored. □

We now proceed as in Sect. 2 using the same concepts. Lemma 4 and the fact that $|I'| \geq 3n^2$ allow us to prove:

Theorem 4. Maximum Clique Partition on Visibility Graph with Holes *cannot be approximated by any polynomial time algorithm with an approximation ratio of* $\frac{|I'|^{1/14-\gamma}}{4}$*, where* $|I'|$ *is the number of vertices in the polygon and where* $\gamma > 0$*, unless* $NP = P$*.*

6 Conclusion

We have studied the two problems Maximum Clique on Visibility Graph and Minimum Clique Partition on Visibility Graph for both polygons with and without holes. In the case of polygons without holes, the clique problem can be solved in polynomial time and this algorithm can be used in an approximation algorithm for the clique partition problem to achieve a logarithmic approximation ratio. The best inapproximability result known for the clique partition problem without holes is APX-hardness, thus the approximability of this problem is not yet precisely characterized.

In the case of polygons with holes, we have shown for both problems inapproximability ratios of n^{ϵ} for some $\epsilon > 0$, and have thus placed these two problems in the corresponding inapproximability class as defined in [2].

Our approach of "characterizing" the class of visibility graphs by studying classic graph problems for this class has been used before – at least implicitly. The computational complexity of the related problem of coloring the vertices of a visibility graph with a minimum number of colors is completely unknown and an open problem for future research [22]. Other open problems include, of course, determining the exact approximation threshold for Minimum Clique Partition on Visibility Graph without Holes.

References

1. A. Aggarwal, S. Ghosh, and R. Shyamasundar; *Computational Complexity of Restricted Polygon Decompositions*; Computational Morphology, G. Toussaint, ed., North-Holland, pp. 1-11, 1988.
2. S. Arora and C. Lund; *Hardness of Approximations*; in: Approximation Algorithms for NP-Hard Problems (ed. Dorit Hochbaum), PWS Publishing Company, pp. 399-446, 1996.
3. D. Avis and D. Rappaport; *Computing the largest empty convex subset of a set of points*; Proc. 1st Ann. ACM Symposium Computational Geometry, pp. 161 - 167, 1985.
4. P. Crescenzi, V. Kann; *A Compendium of NP Optimization Problems*; in the book by G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*, Springer-Verlag, Berlin, 1999; also available in an online-version at: http://www.nada.kth.se/theory/compendium/.
5. J. C. Culberson and R. A. Reckhow; *Covering Polygons is hard*; Proc. 29th Symposium on Foundations of Computer Science, 1988.
6. David P. Dobkin, Herbert Edelsbrunner, and Mark H. Overmars; *Searching for Empty Convex Polygons*; Algorithmica, 5, pp. 561 – 571, 1990.

7. Herbert Edelsbrunner, L. Guibas; *Topologically sweeping an arrangement*; J. Comput. System Sci. 38, pp. 165 – 194, 1989.
8. S. Eidenbenz and P. Widmayer; *An Approximation Algorithm for Minimum Convex Cover with Logarithmic Performance Guarantee*; manuscript, to appear.
9. S. Eidenbenz, C. Stamm, and P. Widmayer; *Inapproximability of some Art Gallery Problems*; Proc. 10th Canadian Conf. Computational Geometry, pp. 64-65, 1998.
10. S. Eidenbenz, C. Stamm, and P. Widmayer; *Positioning Guards at Fixed Height above a Terrain – An Optimum Inapproximability Result*; Lecture Notes in Computer Science, Vol. 1461 (ESA'98), pp. 187-198, 1998.
11. S. Eidenbenz; *Inapproximability Results for Guarding Polygons without Holes*; Lecture Notes in Computer Science, Vol. 1533 (ISAAC'98), pp. 427-436, 1998.
12. S. Eidenbenz; *How Many People Can Hide in a Terrain?*; Lecture Notes in Computer Science 1741 (ISAAC'99), pp. 184-194, 1999.
13. H. Everett, D. Corneil; *Negative Results on Characterizing Visibility Graphs*; Computational Geometry 5, pp. 51-63, 1995.
14. S. Ghosh; *Approximation algorithms for Art Gallery Problems*; Proc. of the Canadian Information Processing Society Congress, 1987.
15. S. Ghosh; *On Recognizing and Characterizing Visibility Graphs of Simple Polygons*; Discrete Comput Geom 17, pp. 143-162, 1997.
16. J. Hastad; *Clique is hard to approximate within* $n^{1-\epsilon}$; Proc. of the Symposium on Foundations of Computer Science, 1996.
17. J. Hershberger; *Finding the Visibility Graph of a Polygon in Time Proportional to its Size*; Proc. of the 3rd Annual ACM Symposium on Computational Geometry, Waterloo, pp. 11 – 20, 1987.
18. D. Hochbaum; *Approximating Covering and Packing Problems: Set Cover, Vertex Cover, Independent Set, and Related Problems*; in: Approximation Algorithms for NP-Hard Problems (ed. Dorit Hochbaum), PWS Publishing Company, pp. 94-143, 1996.
19. D. T. Lee and A. K. Lin; *Computational Complexity of Art Gallery Problems*; IEEE Trans. Info. Th, pp. 276-282, IT-32, 1986.
20. J. O'Rourke and K. J. Supowit; *Some NP-hard Polygon Decomposition Problems*; IEEE Transactions on Information Theory, Vol IT-29, No. 2, 1983.
21. J. O'Rourke; *Art Gallery Theorems and Algorithms*; Oxford University Press, New York (1987).
22. J. O'Rourke; *Open problems in the combinatorics of visibility and illumination*; in Advances in Discrete and Computational Geometry, eds. B. Chazelle and J. E. Goodman and R. Pollack, (Contemporary Mathematics) American Mathematical Society, Providence, pp. 237-243, 1998.
23. C.H. Papadimitriou and M. Yannakakis; *Optimization, approximation, and complexity classes*; Proc. 20th ACM Symposium on the Theory of Computing, pp. 229 - 234, 1988.
24. L. Prasad, S.S. Iyengar; *A note on the combinatorial structure of the visibility graph in simple polygons*; Theoretical Computer Science 140, pp. 249-263, 1995.
25. T. Shermer; *Recent results in Art Galleries*; Proc. of the IEEE, 1992.
26. T. Shermer; *Hiding People in Polygons*; Comuting 42, pp. 109-131, 1989.
27. J. Urrutia; *Art gallery and Illumination Problems*; in Handbook on Computational Geometry, edited by J.-R. Sack and J. Urrutia, 1998.

Real-Time Language Recognition by Alternating Cellular Automata

Thomas Buchholz, Andreas Klein, and Martin Kutrib

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Abstract The capabilities of alternating cellular automata (ACA) to accept formal languages are investigated. Several notions of alternation in cellular automata have been proposed. Here we study so-called nonuniform ACAs. Our investigations center on space bounded real-time computations. In particular, we prove that there is no difference in acceptance power regardless of whether one-way or two-way communication lines are provided. Moreover, the relations between real-time ACAs and deterministic (CA) and nondeterministic (NCA) cellular automata are investigated. It is proved that even the real-time ACAs gain exponential speed-up against nondeterministic NCAs. Comparing ACAs with deterministic CAs it is shown that real-time ACAs are strictly more powerful than real-time CAs.

1 Introduction

Linear arrays of finite automata can be regarded as models for massively parallel computers. Mainly they differ in how the automata are interconnected and in how the input is supplied. Here we are investigating arrays with two very simple interconnection patterns. Each node is connected to its both immediate neighbors or to its right immediate neighbor only. Correspondingly they are said to have two-way or one-way communication lines. The input mode is parallel. At initial time each automaton fetches an input symbol. Such arrays are commonly called cellular automata.

Although deterministic, nondeterministic and alternating finite automata have the same computing capability there appear to be essential differences when they are used to construct deterministic (CA), nondeterministic (NCA) and alternating (ACA) cellular automata. (We use the denotation OCA, NOCA and AOCA to indicate one-way communication lines.) For example, it is a famous open problem whether or not CAs and OCAs have the same computing power (L(OCA) =? L(CA)) [13] but the problem is solved for nondeterministic arrays (L(NOCA) = L(NCA)) [4]. It is known that the real-time OCA languages are properly contained in the linear-time OCA languages ($\mathrm{L}_{rt}(\mathrm{OCA}) \subset \mathrm{L}_{lt}(\mathrm{OCA})$) [3,15,6]. But on the other hand, $\mathrm{L}_{rt}(\mathrm{NOCA}) = \mathrm{L}_{lt}(\mathrm{NOCA})$ has been shown in [1]. Since $\mathrm{L}_{lt}(\mathrm{NOCA}) = \mathrm{L}_{rt}(\mathrm{NCA})$ (which follows from the closure of $\mathrm{L}_{rt}(\mathrm{NOCA})$ under reversal [1] and $\mathrm{L}_{lt}(\mathrm{NOCA}) = \mathrm{L}^{R}_{rt}(\mathrm{NCA})$) we have the identity

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 213–225, 2000.
© Springer-Verlag Berlin Heidelberg 2000

$L_{rt}(NOCA) = L_{rt}(NCA)$. For deterministic arrays it holds $L_{rt}(OCA) \subset L_{rt}(CA)$ [13].

Altogether there is little known about the properness of the known inclusions. The dilemma is emphazised by the open problem whether or not the real-time deterministic CA languages are strictly included in the exponential-time nondeterministic CA languages ($L_{rt}(CA) =? L(NCA)$)! The latter family is identical to NSPACE(n) (the context-sensitive languages), whereas the former is characterizable by one-way two-head alternating finite automata [7].

In order to prove a proper superclass that is as small as possible we cannot add more time but we can strengthen the single cells and simultaneously reduce the time to real-time again.

Therefore, we consider arrays built by alternating finite automata. In [9] from the point of view of time-varying cellular automata first results concerning a restricted variant of ACAs are shown. In a second work on alternating cellular automata [10] three models are distinguished. In *nonuniform* ACAs each cell computes its next state independently according to the local transition function. In *uniform* ACAs at every time step one deterministic local transition is nondeterministically chosen from a finite set of such functions and is applied to all the cells. The last notion defines the *weak* ACAs where only the leftmost cell of the array is an alternating automaton; all the others are nondeterministic. In [10] it is shown that nonuniform ACAs are the most powerful of the devices and that linear-time weak and uniform ACAs coincide. Some other results deal with simulations between alternating Turing machines and ACAs. This topic is also the main contribution of [12] where the simulation results of [10] are extended and some others are shown.

Our main interest are nonuniform ACAs under real-time restriction. The basic notions are defined in the next section. Section 3 is devoted to the question whether or not two-way ACAs are more powerful than one-way AOCAs. We prove the answer to be 'no'. Especially, the equivalence between ACAs and AOCAs is shown for all time complexities. A second result in Section 3 is the important technical lemma which states that a specific subclass of ACAs can be sped up by a constant factor as long as the time complexity does not fall below real-time. For such devices, especially, the equivalence of real-time and linear-time follows. In Section 4 the relations between real-time ACAs and deterministic and nondeterministic cellular automata are investigated. It is proved that even the real-time ACAs gain exponential speed-up against nondeterministic NCAs. Comparing ACAs with deterministic CAs it is shown that real-time ACAs are strictly more powerful than real-time CAs. Thus, a proper superclass of the real-time CA languages is obtained. Since NSPACE(n) is included in ATIME(n^2) and, on the other hand, $L_{rt}(ACA)$ will be shown to contain NSPACE(n) and is contained in ATIME(n^2) either [10] we conclude that the real-time ACAs are a reasonable model at all.

The latter result becomes important in so far as it is not known whether one of the following inclusions is strict:

$$L_{rt}(CA) \subseteq L_{lt}(CA) \subseteq L(OCA) \subseteq L(CA) \subseteq L(NCA)$$

2 Basic Notions

We denote the rational numbers by $\mathbb{Q}$, the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \ldots\}$ by $\mathbb{N}$, the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$ and the powerset of a set S by 2^S. The empty word is denoted by ε and the reversal of a word w by w^R. For the length of w we write $|w|$.

An alternating cellular automaton is a linear array of identical alternating finite automata, sometimes called cells, where each of them is connected to its both nearest neighbors (one to the left and one to the right). For our convenience we identify the cells by positive integers. The state transition of the cells depends on the actual state of the cell itself and the actual states of its both neighbors. The finite automata work synchronously at discrete time steps. Their states are partitioned into existential and universal ones. What makes a, so far, nondeterministic computation to an alternating computation is the mode of acceptance, which will be defined with respect to the partitioning. More formally:

Definition 1.
An alternating cellular automaton *(ACA) is a system* $(S, \delta, \#, A, F)$ *where*

1. S *is the finite, nonempty set of* states *which is partitioned into* existential *(*S_e*) and* universal *(*S_u*) states:* $S = S_e \cup S_u$,
2. $\# \notin S$ *is the* boundary state,
3. $A \subseteq S$ *is the nonempty set of* input symbols,
4. $F \subseteq S$ *is the set of* accepting states,
5. δ *is the finite, nonempty set of* local transition functions *which map from* $\big(S \cup \{\#\}\big)^3$ *to* S.

Let $\mathcal{M} = (S, \delta, \#, A, F)$ be an ACA. A *configuration* of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state, which is actually a mapping $c_t : [1, \ldots, n] \to S$ for $n \in \mathbb{N}$. The configuration at time 0 is defined by the initial sequence of states. For a given input word $w = w_1 \cdots w_n \in A^+$ we set $c_{0,w}(i) := w_i$, $1 \leq i \leq n$. Subsequent configurations are chosen according to the global transition Δ:

Let $n \in \mathbb{N}$ be a positive integer and c resp. c' be two configurations defined by $s_1, \ldots, s_n \in S$ resp. $s'_1, \ldots, s'_n \in S$.

$$c' \in \Delta(c) \iff \exists\, \delta_1, \ldots, \delta_n \in \delta : \\ s'_1 = \delta_1(\#, s_1, s_2), s'_2 = \delta_2(s_1, s_2, s_3), \ldots, s'_n = \delta_n(s_{n-1}, s_n, \#)$$

Thus, Δ is induced by δ. Observe, that one can equivalently define ACAs by requiring just one unique nondeterministic local transition that maps from $\big(S \cup \{\#\}\big)^3$ to $\big(2^S \setminus \emptyset\big)$. But with an eye towards later constructions we are requiring a finite, nonempty set of deterministic local transitions from which each cell nondeterministically chooses one at every time step. Obviously, both definitions yield equivalent devices.

The evolution of $\mathcal{M}$ is represented by its computation tree.

The *computation tree* $T_{\mathcal{M},w}$ of $\mathcal{M}$ under input $w \in A^+$ is a tree whose nodes are labeled by configurations. The root of $T_{\mathcal{M},w}$ is labeled by $c_{0,w}$. The children

of a node labeled by a configuration c are the nodes labeled by the possible successor configurations of c. Thus, the node c has exactly $|\Delta(c)|$ children.

If the state set is a Cartesian product of some smaller sets $S = S_0 \times S_1 \times \cdots \times S_r$, we will use the notion *register* for the single parts of a state. The concatenation of a specific register of all cells forms a *track*.

If the flow of information is restricted to one-way, the resulting device is an *alternating one-way cellular automaton* (AOCA). I.e. the next state of each cell depends on the actual state of the cell itself and the state of its immediate neighbor to the right. Thus, we have information flow from right to left. Accordingly acceptance in ACAs and AOCAs is indicated by the leftmost cell of the array: A *configuration c is accepting* iff $c(1) \in F$.

In order to define *accepting computations* on input words we need the notion of accepting subtrees.

Definition 2. *Let $\mathcal{M} = (S, \delta, \#, A, F)$ be an ACA or an* AOCA *and $T_{\mathcal{M},w}$ be its computation tree for an input word $w \in A^n$, $n \in \mathbb{N}$. A finite subtree T' of $T_{\mathcal{M},w}$ is said to be an* accepting subtree *iff it fulfills the following conditions:*

1. *The root of T' is the root of $T_{\mathcal{M},w}$.*
2. *Let c be a node in T'. If $c' \in \Delta(c)$ is a child of c in T' then the set of all children of c in T' is $\{c'' \in \Delta(c) \mid c''(i) = c'(i)$ for all $1 \leq i \leq n$ such that $c(i) \in S_e\}$.*
3. *The leafs of T' are labeled by accepting configurations.*

From the computational point of view an accepting subtree is built by letting all the cells in existential states do their nondeterministic guesses and, subsequently, spawning all possible distinct offspring configurations with respect to the cells in universal states.

Conversely, one could build the subtree by spawning all possible distinct offspring configurations with respect to the cells in universal states at first, and letting cells in existential states do their guesses in each offspring configuration independently. Fortunately, it has been shown [12] that both methods lead to time complexities which differ at most by a constant factor. Moreover, the proofs given in the following can easily be adapted to that mode of acceptance such that both methods are equivalent in the framework in question.

Definition 3. *Let $\mathcal{M} = (S, \delta, \#, A, F)$ be an ACA or an* AOCA.

1. *A word $w \in A^+$ is* accepted *by $\mathcal{M}$ if there exists an accepting subtree of $T_{\mathcal{M},w}$.*
2. *$L(\mathcal{M}) = \{w \in A^+ \mid w$ is accepted by $\mathcal{M}\}$ is the* language accepted *by $\mathcal{M}$.*
3. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. If for all $w \in L(\mathcal{M})$ there exists an accepting subtree of $T_{\mathcal{M},w}$ the height of which is less than $t(|w|)$, then L is said to be of* time complexity *t.*

An ACA (AOCA) $\mathcal{M}$ is *nondeterministic* if the state set consists of existential states only. An accepting subtree is now a list of configurations which corresponds

to a possible computation path of $\mathcal{M}$. Nondeterministic cellular automata are denoted by NCA resp. NOCA.

An ACA (AOCA) is *deterministic* if the set δ of local transition functions is a singleton. In these cases the course of computation is unique for a given input word w and, thus, the whole computation tree is a list of configurations. Deterministic cellular automata are denoted by CA resp. OCA.

The family of all languages which can be accepted by devices of a type POLY with time complexity t is denoted by $\mathrm{L}_t(\mathrm{POLY})$. If t equals the *identity function* $id(n) := n$ acceptance is said to be in real-time and we write $\mathrm{L}_{rt}(\mathrm{POLY})$. The *linear-time* languages $\mathrm{L}_{lt}(\mathrm{POLY})$ are defined according to

$$\mathrm{L}_{lt}(\mathrm{POLY}) := \bigcup_{k\in\mathbb{Q},\, k\geq 1} \mathrm{L}_{k\cdot id}(\mathrm{POLY})$$

3 Equivalence of One-Way and Two-Way Information Flow and Linear Speed-up

This section is devoted to the relationship between ACAs and AOCAs and the speed-up of a restricted version that becomes important in subsequent proofs. The main results are that for arbitrary time complexities there is no difference in acceptance power between one-way and two-way information flow and the possibility to speed up so-called uniformly universal ACAs and AOCAs by a constant factor as long as they do not fall below real-time. Especially by the latter result we can show the results in the next sections even for real-time language families.

Without loss of generality we may assume that once a cell becomes accepting it remains in accepting states permanently. Such a behavior is simply implemented by setting a flag in an additional register that will never be unset. Obviously, thereby the accepted language is not affected since if an node labeled by an accepting configuration belongs to a finite accepting subtree then there exists a finite accepting subtree where the node is a leaf (it is simply constructed by omitting all offsprings of that node).

The next result states that one-way information flow in alternating cellular automata is as powerful as two-way information flow. This, on one hand, gives us a normalization since for proofs and constructions it is often useful to reduce the technical challenge to one-way transitions and, on the other hand, indicates the power of alternations since it is well known that deterministic one-way languages form a proper subset of the deterministic two-way languages: $\mathrm{L}_{rt}(\mathrm{OCA}) \subset \mathrm{L}_{rt}(\mathrm{CA})$ [13].

Theorem 4. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping. Then*

$$\mathrm{L}_t(\mathrm{AOCA}) = \mathrm{L}_t(\mathrm{ACA})$$

Proof. For structural reasons it suffices to show $\mathrm{L}_t(\mathrm{ACA}) \subseteq \mathrm{L}_t(\mathrm{AOCA})$.

The idea for the simulation of an ACA by an AOCA without any loss of time is as follows: A cell of the AOCA 'knows' the actual states of itself and of its neighbor to the right. Additionally, it guesses the state of its neighbor to the left and simulates the two-way transition of the ACA. In order to verify whether or not the guesses are correct each cell stores its guessed state and its old state in additional registers. After performing a simulation step the verification can simply be done by comparing the old state of a cell with the guessed state of its neighbor to the right. Thus, the verification is done by the neighbor to the left of a cell, respectively.

Obviously, the guesses of the leftmost cell are not verified. But we can restrict the local transition as follows: If the initial state of a cell is existential and its guessed left neighbor state is not the border state then it is marked by a '−' during the first time step. If the initial state of a cell is universal and its guessed left neighbor state is not the border state then it is marked by a '+'. The effect of these marks is that the cells with a '−' will never and the cells with a '+' will always accept. Thus, if the cell is not the leftmost cell this behavior does not affect the overall computation result. But if the cell is the leftmost cell only the correct guesses are relevant during the remaining computation.

Moreover, a left border state is guessed by a cell if and only if that cell has guessed a left border state at the first time step. Therefore, to guess a left border state at every time step is the only way for the leftmost cell to become accepting. But exactly in these cases it has simulated the correct behavior of the leftmost cell of the two-way ACA.

Up to now we kept quiet about a crucial point. Whereas the verification itself is a deterministic task which can be performed by cells in existential as well as in universal states, responding to the result of the verification needs further mechanisms.

We distinguish two cases: If the old state of a cell is an existential one and the verification by the left neighboring cell fails then the latter sends an error signal to the left that prevents the not marked cells passed through from accepting. Therefore, in an accepting subtree there are only nodes labeled by configurations in which existential cells have guessed right and, hence, have simulated the two-way transition correctly. If the verification succeeds no further reaction is necessary.

In the second case the old state of a cell is an universal one. If the verification by the left neighboring cell fails it sends an error signal to the left that enforces all not marked cells passed through to switch into an accepting state. Again, if the verification succeeds no further reaction is necessary.

What is the effect of these mechanisms: In an accepting subtree in all configurations with a common predecessor cells that have been existential in the predecessor are in the same states, respectively. Due to the first case these cells have simulated the two-way transition correctly. Since all siblings (spawned by universal states) have to lead to subtrees with accepting leafs but acceptance according to the two-way ACA depends on the configurations with correct guesses

only, all configurations with wrong guesses are forced to accept to achieve the desired behavior.

Altogether it follows that the AOCA can simulate the ACA without any loss of time. □

Corollary 5. $\mathrm{L}_{rt}(\mathrm{AOCA}) = \mathrm{L}_{rt}(\mathrm{ACA})$

As we have shown extending the information flow from one-way to two-way does not lead to more powerful devices. The next lemma states that increasing the computation time by a constant factor does not either if we restrict the computations to the uniformly universal mode. A corresponding result does not hold for deterministic cellular automata. Instead, $\mathrm{L}_{rt}(\mathrm{OCA}) \subset \mathrm{L}_{lt}(\mathrm{OCA})$ has been shown [3,6,15]. The relationship is a famous open problem for deterministic two-way devices (e.g. [5,14]).

Uniform ACAs have been introduced in [9,10]. The main difference between uniform ACAs and (nonuniform) ACAs is the induction of the global transition. Whereas in an ACA at every time step each cell chooses independently one local transition, in an uniform ACA at every time step one local transition is chosen globally and applied to all the cells:

Let $\mathcal{M} = (S, \delta, \#, A, F)$ be an uniform ACA, $n \in \mathbb{N}$ be a positive integer and c resp. c' be two configurations defined by $s_1, \ldots, s_n \in S$ resp. $s'_1, \ldots, s'_n \in S$.

$$c' \in \Delta(c) \iff \exists\, \delta_u \in \delta : \\ s'_1 = \delta_u(\#, s_1, s_2), s'_2 = \delta_u(s_1, s_2, s_3), \ldots, s'_n = \delta_u(s_{n-1}, s_n, \#)$$

Thus, in a computation tree of an uniform ACA each node has at most $|\delta|$ successors. Now a whole configuration is labeled universal (existential) if the leftmost cell is in an universal (existential) state. An accepting subtree is a finite subtree of the computation tree that includes all (one) of the successors of a universal (existential) node. As usual all leafs have to be labeled with accepting configurations.

Now we are combining both modes in order to define an intermediate model that serves in later proofs as a helpful tool since its time complexity can be reduced by a constant factor. We are considering a computation mode that is nonuniform for existential and uniform for universal states. It is called *uniformly universal* mode and the corresponding devices are denoted by UUACA and UUAOCA.

For our purposes it is sufficient to consider UUAOCAs $\mathcal{M} = (S, \delta, \#, A, F)$ which are *alternation normalized* as follows:

$$A \subseteq S_e \text{ and } \forall\, \delta_i \in \delta : \\ \big(\forall\, s_1 \in S_e, s_2 \in S_e \cup \{\#\} : \delta_i(s_1, s_2) \in S_u \text{ and} \\ \forall\, s_1 \in S_u, s_2 \in S_u \cup \{\#\} : \delta_i(s_1, s_2) \in S_e\big)$$

Thus, at every even time step all the cells are in existential and at every odd time step all the cells are in universal states.

Lemma 6. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping and $k \in \mathbb{Q}$, $k \geq 1$, be a constant. For every alternation normalized* UUAOCA $\mathcal{M}$ *that accepts a language $L(\mathcal{M})$ with time complexity $k \cdot t$ there exists an alternation normalized* UUAOCA $\mathcal{M}'$ *with time complexity t such that $L(\mathcal{M}) = L(\mathcal{M}')$ and vice versa.*

One central point of the proof is that the number of successor configurations of an universal configuration in uniformly universal mode is bounded by $|\delta|$. Its details can be found in [2].

Corollary 7. *Let $t : \mathbb{N} \to \mathbb{N}$, $t(n) \geq n$, be a mapping and $k \in \mathbb{Q}$, $k \geq 1$, be a constant. For every alternation normalized* UUACA $\mathcal{M}$ *that accepts a language $L(\mathcal{M})$ with time complexity $k \cdot t$ there exists an alternation normalized* UUACA $\mathcal{M}'$ *with time complexity t such that $L(\mathcal{M}) = L(\mathcal{M}')$ and vice versa.*

Proof. The construction of Theorem 4 does not affect the status of the cells (i.e. whether they are existential or universal). Therefore, for a given alternation normalized UUACA there exists an equivalent alternation normalized UUAOCA with the same time complexity. The UUAOCA can be sped up and the resulting automaton, trivially, can be transformed into an alternation normalized UUACA again. □

4 Comparisons with (Non)Deterministic Cellular Automata

It is a famous open problem whether or not the inclusion $\mathrm{L}_{rt}(\mathrm{CA}) \subseteq ? \mathrm{L}_{lt}(\mathrm{CA})$ is a proper one. Moreover, the seemingly easier problem $\mathrm{L}_{rt}(\mathrm{CA}) \subseteq ? \mathrm{L}(\mathrm{CA})$ is open, too. The same holds for nondeterministic cellular automata: It is not known whether or not the inclusion $\mathrm{L}_{rt}(\mathrm{NCA}) \subseteq ? \mathrm{L}(\mathrm{NCA})$ is strict.

Since L(NCA) coincides with the context-sensitive languages and L(CA) with the deterministic context-sensitive languages the properness of the inclusion $\mathrm{L}_{rt}(\mathrm{CA}) \subseteq ? \mathrm{L}(\mathrm{NCA})$ is also open due to the open problems mentioned and the famous lba-problem (i.e. in our terms $\mathrm{L}(\mathrm{CA}) = ? \mathrm{L}(\mathrm{NCA})$). The open problem $\mathrm{L}_{rt}(\mathrm{CA}) \subseteq ? \mathrm{L}_{rt}(\mathrm{NCA})$ stresses the dilemma. Altogether, the following inclusions follow for structural reasons but it is not known whether one of them is strict.

$$\mathrm{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{L}(\mathrm{OCA}) \subseteq \mathrm{L}(\mathrm{CA}) \subseteq \mathrm{L}(\mathrm{NCA}) \quad \text{and}$$
$$\mathrm{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{L}_{rt}(\mathrm{NCA}) \subseteq \mathrm{L}(\mathrm{NCA}).$$

In the present section we compare real-time ACAs to deterministic and nondeterministic cellular automata. The next result shows that adding alternations to nondeterministic computations yields enormous speed-ups.

Theorem 8. $\mathrm{L}(\mathrm{NCA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA})$

Proof. Let $\mathcal{M} = (S, \delta, \#, A, F)$ be an NCA. Since the number of cells is bounded by the length of the input n the number of configurations is bounded by $|S|^n$. Therefore, we can assume that $\mathcal{M}$ has the exponential time complexity $|S|^n$.

The following construction is done for $|S| = 2$. A generalization is straightforward. The first step is to define an alternation normalized UUACA $\mathcal{M}' = (S', \delta', \#, A, F')$ which needs $3 \cdot n$ time steps to simulate 2^n steps of the NCA $\mathcal{M}$. For the main part of $\mathcal{M}'$ (a deterministic track is added later) we set

$S' := A \cup S^2 \cup S^3, \quad S'_e := A \cup S^2, \quad S'_u := S^3,$

$\delta' := \{\delta^e_{i,j,k} \mid 1 \le i,j \le |S|, 1 \le k \le |F|\} \cup \{\delta^e_i \mid 1 \le i \le |S|\} \cup \{\delta^u_1, \delta^u_2\},$

where $\delta^e_{i,j,k}$, δ^e_i, δ^u_1 and δ^u_2 will be defined in the following.

Let n denote the length of the input, $\delta = \{\delta_1, \ldots, \delta_d\}$, $S = \{s_1, \ldots, s_r\}$ and $F = \{f_1, \ldots, f_v\}$.

In its first time step $\mathcal{M}'$ saves its input configuration, guesses an accepting configuration of $\mathcal{M}$ and another configuration of $\mathcal{M}$:

$\forall\, p_1, p_2 \in A, p_3 \in A \cup \{\#\}$:

$\delta^e_{i,j,k}(p_1, p_2, p_3) := (p_2, s_i, s_j)$

$\delta^e_{i,j,k}(\#, p_2, p_3) := (p_2, s_i, f_k), \quad 1 \le i,j \le |S|, \quad 1 \le k \le |F|$

The idea is to guess successively an accepting computation path $c_0, \ldots, c_{2^n}$ of $\mathcal{M}$. The configuration on the second track should be $c_{2^{n-1}}$ and the accepting configuration on the third track should be c_{2^n}.

From now on at every universal step for each configuration two offsprings are spawned. One gets the configurations on the first and second track and the other the configurations on the second and third track:

$\forall\, (p_{1,1}, p_{1,2}, p_{1,3}), (p_{3,1}, p_{3,2}, p_{3,3}) \in S^3 \cup \{\#\}, (p_{2,1}, p_{2,2}, p_{2,3}) \in S^3$:

$\delta^u_1\big((p_{1,1}, p_{1,2}, p_{1,3}), (p_{2,1}, p_{2,2}, p_{2,3}), (p_{3,1}, p_{3,2}, p_{3,3})\big) := (p_{2,1}, p_{2,2})$

$\delta^u_2\big((p_{1,1}, p_{1,2}, p_{1,3}), (p_{2,1}, p_{2,2}, p_{2,3}), (p_{3,1}, p_{3,2}, p_{3,3})\big) := (p_{2,2}, p_{2,3})$

Since c'_1 represents the configurations c_0, $c_{2^{n-1}}$ and c_{2^n} (on its first, second and third track) its both successors represent the configuration pairs $(c_0, c_{2^{n-1}})$ and $(c_{2^{n-1}}, c_{2^n})$. (The notation (c_i, c_j) says that the first track contains c_i and the second one c_j.)

In every further existential step the configuration between the represented configurations is guessed:

$\forall\, (p_{1,1}, p_{1,2}), (p_{3,1}, p_{3,2}) \in S^2 \cup \{\#\}, (p_{2,1}, p_{2,2}) \in S^2$:

$\delta^e_i\big((p_{1,1}, p_{1,2}), (p_{2,1}, p_{2,2}), (p_{3,1}, p_{3,2})\big) := (p_{2,1}, s_i, p_{2,2}), \quad 1 \le i \le |S|$

Thus, the two possible configurations of $\mathcal{M}'$ at time step 3 are representing the configuration triples $(c_0, c_{2^{n-2}}, c_{2 \cdot 2^{n-2}})$ and $(c_{2 \cdot 2^{n-2}}, c_{3 \cdot 2^{n-2}}, c_{4 \cdot 2^{n-2}})$. One time step later we have the four pairs $(c_0, c_{2^{n-2}})$, $(c_{2^{n-2}}, c_{2 \cdot 2^{n-2}})$, $(c_{2 \cdot 2^{n-2}}, c_{3 \cdot 2^{n-2}})$ and $(c_{3 \cdot 2^{n-2}}, c_{4 \cdot 2^{n-2}})$.

Concluding inductively it is easy to see that at time $t = 2 \cdot m$, $1 \le m \le n$ there exist 2^t configurations of $\mathcal{M}'$ representing the pairs $(c_0, c_{2^{n-t}})$, $(c_{2^{n-t}}, c_{2 \cdot 2^{n-t}})$, $(c_{2 \cdot 2^{n-t}}, c_{3 \cdot 2^{n-t}}), \ldots, (c_{(2^t-1) \cdot 2^{n-t}}, c_{2^t \cdot 2^{n-t}})$.

For $t = 2 \cdot n$ we obtain $(c_0, c_1), (c_1, c_2), \ldots, (c_{2^n-1}, c_{2^n})$. Now $\mathcal{M}'$ can locally check whether the second element of a pair is a valid successor of the first elements of the cell and its neighbors according to the local transitions of $\mathcal{M}$. If the check succeeds $\mathcal{M}'$ has guessed an accepting computation path of $\mathcal{M}$ and accepts the input.

In order to perform the check each cell of $\mathcal{M}'$ has to be aware of the time step $2 \cdot n$. For this purpose a deterministic FSSP algorithm [11,16] is started on an additional track which synchronizes the cells at time $2 \cdot n$. Altogether the result of the check is available at time step $2 \cdot n + 1$ and needs another $n - 1$ time steps to get into the leftmost cell. We conclude that $\mathcal{M}'$ has time complexity $3 \cdot n$. By Lemma 6 the alternation normalized UUACA $\mathcal{M}'$ can be sped up to real-time.

It remains to show that $\mathrm{L}_{rt}(\mathrm{UUACA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA})$. The proof is a straightforward adaption of the proof that (nonuniform) ACAs are at least as powerful as uniform ACAs [10]. □

Corollary 9. $\mathrm{L}(\mathrm{NCA}) \subseteq \mathrm{L}_{rt}(\mathrm{AOCA})$

Extending the previously mentioned chains of inclusions by the last result we obtain

$$\mathrm{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{L}(\mathrm{OCA}) \subseteq \mathrm{L}(\mathrm{CA}) \subseteq \mathrm{L}(\mathrm{NCA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA}) \quad \text{and}$$
$$\mathrm{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{L}_{rt}(\mathrm{NCA}) \subseteq \mathrm{L}(\mathrm{NCA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA}).$$

The next result shows that in both chains one of the inclusions *is* a proper one. It states $\mathrm{L}_{rt}(\mathrm{CA}) \subset \mathrm{L}_{rt}(\mathrm{ACA})$. We prove the inclusion by the use of a specific kind of deterministic cellular spaces as connecting pieces. A *deterministic cellular space* (CS) works like a deterministic cellular automaton. The difference is the unbounded number of cells. In cellular spaces there exists a so-called *quiescent state* q_0 such that the local transition satisfies $\delta(q_0, q_0, q_0) = q_0$. At time 0 all the cells from $\mathbb{Z}$ except the cells $1, \ldots, n$ which get the input are in the quiescent state. Obviously, at every time step the number of nonquiescent cells increases at most by 2.

In [8] an infinite hierarchy of language families has been shown: For example, if $r \in \mathbb{Q}$, $r \geq 1$, and $\varepsilon \in \mathbb{Q}$, $\varepsilon > 0$, then $\mathrm{L}_{n^r}(\mathrm{CS}) \subset \mathrm{L}_{n^{r+\varepsilon}}(\mathrm{CS})$.

Especially, for $r = 1$ and $\varepsilon = 1$ it holds $\mathrm{L}_{rt}(\mathrm{CS}) \subset \mathrm{L}_{n^2}(\mathrm{CS})$.

Cellular spaces which are bounded to the left (and unbounded to the right) are equivalent to the original model since both halflines can be simulated on different tracks in parallel. Moreover, one obtains again an equivalent model if the number of cells is bounded by the time complexity. Let $w = w_1 \cdots w_n$ be an input word and $s : \mathbb{N} \to \mathbb{N}$, $s(n) \geq n$, be a mapping. The family of languages acceptable by deterministic cellular automata with initial configuration

$$c_{0,w} : [1, \ldots, s(|w|)] \to S, \quad c_{0,w} = \begin{cases} w_i \text{ if } 1 \leq i \leq |w| \\ q_0 \text{ if } |w| + 1 \leq i \leq s(|w|) \end{cases}$$

is denoted by $\mathrm{L}_{t,s}(\mathrm{CA})$. It follows immediately $\mathrm{L}_t(\mathrm{CS}) = \mathrm{L}_{t,t}(\mathrm{CA})$ (here we assume always $\delta(q_0, q_0, \#) = q_0$).

Theorem 10. $\mathrm{L}_{rt}(\mathrm{CA}) \subset \mathrm{L}_{rt}(\mathrm{ACA})$

Proof. From $\mathrm{L}_t(\mathrm{CS}) = \mathrm{L}_{t,t}(\mathrm{CA})$ for $t = id$ we obtain $\mathrm{L}_{rt}(\mathrm{CS}) = \mathrm{L}_{n,n}(\mathrm{CA})$. The latter family is based on simultaneously n-time bounded and n-space bounded cellular automata, i.e. real-time (classical) cellular automata. Thus, $\mathrm{L}_{rt}(\mathrm{CS}) = \mathrm{L}_{n,n}(\mathrm{CA}) = \mathrm{L}_{rt}(\mathrm{CA})$. By the result in [8] for $r = 1$ and $\varepsilon = 1$ it follows $\mathrm{L}_{rt}(\mathrm{CS}) \subset \mathrm{L}_{n^2}(\mathrm{CS})$ which is equivalent to $\mathrm{L}_{rt}(\mathrm{CA}) \subset \mathrm{L}_{n^2}(\mathrm{CS}) = \mathrm{L}_{n^2,n^2}(\mathrm{CA})$. Now in order to prove the theorem we have to show $\mathrm{L}_{n^2,n^2}(\mathrm{CA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA})$.

The following construction results in an alternation normalized UUACA $\mathcal{M}'$ which simulates a simultaneously n^2-time bounded and n^2-space bounded CA $\mathcal{M}$. Note that a deterministic computation task can per se be regarded as alternation normalized and meets the conditions of the uniformly universal computation mode.

Let $c_0, \ldots, c_{n^2}$ denote the configurations of an accepting computation path of $\mathcal{M}$ on input $w = w_1 \cdots w_n$. $\mathcal{M}'$ gets the input $c'_0(i) = c_0(i) = w_i$, $1 \leq i \leq n$, and 'knows' $c_0(i)$, $n+1 \leq i \leq n^2$, to be the quiescent state q_0.

The key idea for $\mathcal{M}'$ is to guess the states $c_{n^2-n}(1), \ldots, c_{n^2-n}(n)$ existentially during the first time step and subsequently to spawn two offspring computations universally. One of them is the deterministic task to simulate $\mathcal{M}$ on input $c_{n^2-n}(1), \ldots, c_{n^2-n}(n)$ for n time steps in order to check whether $\mathcal{M}$ would accept. The second offspring has to verify whether the guess has been correct (i.e. $\mathcal{M}$ produces a corresponding configuration at time step $n^2 - n$). Therefore, at the third time step it guesses the states $c_{n^2-2\cdot n}(1), \ldots, c_{n^2-2\cdot n}(2 \cdot n)$ two times on three tracks: On one track in the compressed form (i.e. every cell contains two states), on another track the states $c_{n^2-2\cdot n}(1), \ldots, c_{n^2-2\cdot n}(n)$ and on a third track the states $c_{n^2-2\cdot n}(n+1), \ldots, c_{n^2-2\cdot n}(2\cdot n)$. (Whether or not the guess yields two times the same sequence can deterministically be checked.) At the next time step $\mathcal{M}'$ universally spawns three offsprings: One of them is the deterministic task to simulate $\mathcal{M}$ on $c_{n^2-2\cdot n}(1), \ldots, c_{n^2-2\cdot n}(2 \cdot n)$ for n time steps to check whether $\mathcal{M}$ would compute the previously guessed states $c_{n^2-n}(1), \ldots, c_{n^2-n}(n)$ and, thus, to verify the previous guess. The second and third offsprings have to verify whether the new guesses are correct. The second offspring guesses $c_{n^2-3\cdot n}(1), \ldots, c_{n^2-3\cdot n}(2\cdot n)$ and iterates the described procedure. The third task has to guess the states $c_{n^2-3\cdot n}(1), \ldots, c_{n^2-3\cdot n}(3 \cdot n)$ two times at four tracks: In the compressed form (i.e. three states per cell) and $c_{n^2-3\cdot n}(1), \ldots, c_{n^2-3\cdot n}(n)$ and $c_{n^2-3\cdot n}(n+1), \ldots, c_{n^2-3\cdot n}(2 \cdot n)$ and $c_{n^2-3\cdot n}(2n+1), \ldots, c_{n^2-3\cdot n}(3 \cdot n)$ on separate tracks. Now a corresponding procedure is iterated. After the guessing one offspring simulates $\mathcal{M}$ for n time steps on $c_{n^2-3\cdot n}(1), \ldots, c_{n^2-3\cdot n}(3 \cdot n)$ and another three offsprings are verifying the guesses.

Concluding inductively at time $2 \cdot i$ there exist offspring computations for the verification of $c_{n^2-i\cdot n}(j \cdot n + 1), \ldots, c_{n^2-i\cdot n}((j+1) \cdot n)$ where $2 \leq i \leq n$ and $0 \leq j \leq i - 1$.

For $i = n$ the sequences $c_0(j \cdot n + 1), \ldots, c_0((j+1) \cdot n)$ have to be verified. This can be done by checking whether the states match the initial input. For this reason the cells have to be aware of the time step $2 \cdot n$ what can be achieved by providing a deterministic FSSP algorithm on an additional track as has been done in the previous proof. Moreover, the computations have to know to which

initial input symbols their sequences have to be compared. These that verify the sequences $c_{n^2-i\cdot n}(1), \ldots, c_{n^2-i\cdot n}(n)$ behave slightly different. They are spawning three (instead of four) offsprings at every universal step. Since exactly the sequence $c_0(1), \ldots, c_0(n)$ has to be compared to the input $w_1 \cdots w_n$ whereas all other sequences simply have to be compared to $q_0, \ldots, q_0$, the verification can be done.

The FSSP fires at time $2 \cdot n$. Afterwards the (partial) simulation of $\mathcal{M}$ needs another n time steps. To collect the result of that simulation and to get it into the leftmost cell needs at most n further time steps. Altogether, $\mathcal{M}'$ has the time complexity $4 \cdot n$. Following the last steps of the proof of Theorem 8 the alternation normalized UUACA $\mathcal{M}'$ can be sped up to real-time and one can conclude $L(\mathcal{M}') \in \mathrm{L}_{rt}(\mathrm{ACA})$. □

Altogether in the previous construction the number of states of $\mathcal{M}'$ depends linearly on the number of states of $\mathcal{M}$.

Another interpretation of the last theorem is the possibility to save time and space simultaneously when adding alternation to a deterministic computation. Moreover, now we know that at least one of the following inclusions is strict:

$$\mathrm{L}_{rt}(\mathrm{CA}) \subseteq \mathrm{L}_{rt}(\mathrm{NCA}) \subseteq \mathrm{L}_{rt}(\mathrm{ACA})$$

Acknowledgements

The authors are grateful to Thomas Worsch for valuable comments on the proof of Theorem 8.

References

[1] Buchholz, Th., Klein, A., and Kutrib, M. *One guess one-way cellular arrays*. Mathematical Foundations in Computer Science 1998, LNCS 1450, 1998, pp. 807–815.

[2] Buchholz, Th., Klein, A., and Kutrib, M. *Real-time language recognition by alternating cellular automata*. IFIG Research Report 9904, Institute of Informatics, University of Giessen, 1999.

[3] Choffrut, C. and Čulik II, K. *On real-time cellular automata and trellis automata*. Acta Inf. 21 (1984), 393–407.

[4] Dyer, C. R. *One-way bounded cellular automata*. Inform. Control 44 (1980), 261–281.

[5] Ibarra, O. H. and Jiang, T. *Relating the power of cellular arrays to their closure properties*. Theoret. Comput. Sci. 57 (1988), 225–238.

[6] Ibarra, O. H. and Palis, M. A. *Some results concerning linear iterative (systolic) arrays*. J. Parallel and Distributed Comput. 2 (1985), 182–218.

[7] Ito, A., Inoue, K., and Wang, Y. *Alternating automata characterizations of one-way iterative arrays*. International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, 1999, pp. 119–128.

[8] Iwamoto, C., Hatsuyama, T., Morita, K., and Imai, K. *On time-constructible functions in one-dimensional cellular automata*. Fundamentals of Computation Theory 1999, LNCS 1684, 1999, pp. 317–326.

[9] Krithivasan, K. and Mahajan, M. *Nondeterministic, probabilistic and alternating computations on cellular array models*. Theoret. Comput. Sci. 143 (1995), 23–49.
[10] Matamala, M. *Alternation on cellular automata*. Theoret. Comput. Sci. 180 (1997), 229–241.
[11] Mazoyer, J. *A six-state minimal time solution to the firing squad synchronization problem*. Theoret. Comput. Sci. 50 (1987), 183–238.
[12] Reischle, F. and Worsch, Th. *Simulations between alternating CA, alternating TM and circuit families*. Technical Report 9/98, Fakultät für Informatik, Universität Karlsruhe, 1998.
[13] Seidel, S. R. *Language recognition and the synchronization of cellular automata*. Technical Report 79-02, Department of Computer Science, University of Iowa, Iowa City, 1979.
[14] Smith III, A. R. *Real-time language recognition by one-dimensional cellular automata*. J. Comput. System Sci. 6 (1972), 233–253.
[15] Umeo, H., Morita, K., and Sugata, K. *Deterministic one-way simulation of two-way real-time cellular automata and its related problems*. Inform. Process. Lett. 14 (1982), 158–161.
[16] Waksman, A. *An optimum solution to the firing squad synchronization problem*. Inform. Control 9 (1966), 66–78.

Damage Spreading and μ-Sensitivity on Cellular Automata

Bruno Martin

LIP, ENS de Lyon, 46, allée d'Italie, 69364 Lyon Cedex 07, France,
Bruno.Martin@ens-lyon.fr,
http://www.ens-lyon.fr/~bmartin

Abstract. We show relations between new notions on cellular automata based on topological and measure-theoretical concepts: almost everywhere sensitivity to initial conditions for Besicovitch pseudo-distance, damage spreading (which measures the information (or damage) propagation) and the destruction of the initial configuration information. Through natural examples, we illustrate the links between these formal definitions and Wolfram's empirical classification.

Introduction

A radius-r unidimensional cellular automaton (CA) is an infinite succession of identical finite-state machines (indexed by $\mathbb{Z}$) called cells. Each finite-state machine is in a state and these states change simultaneously according to a local transition function: the following state of the machine is related to its own state as well as the states of its $2r$ neighbors. A configuration of an automaton is the function which associates to each cell a state. We can thus define a global transition function from the set of all the configurations into itself which associates the following configuration after one step of computation.

An evolution of a unidimensional cellular automaton is usually represented by a space-time diagram. Being given an initial configuration, we represent in $\mathbb{Z} \times \mathbb{N}$ the cellular automaton successive configurations.

Recently, a lot of articles proposed classifications of cellular automata [13, 6] but the reference is still Wolfram's empirical classification [15] which has resisted numerous attempts of formalization [14]. The classification of Gilman [7] is interesting because it is not a classification of CAs, but a classification of couples (CA, measure on its configuration set). This choice, not motivated in the paper, seems interesting because we will illustrate on an example that the intuitive Wolfram's classification depends on a measure, that is a way to choose a random configuration. Actually, due to their local interactions, CAs are often used to simulate physics phenomena and many of them, for instance fluid flow, present a non chaotic or chaotic behavior depending on some parameters (here the fluid speed).

Recently, two very different ways have been investigated to find a definition of chaos for cellular automata that fits with our intuition. On the one hand,

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 226–241, 2000.
© Springer-Verlag Berlin Heidelberg 2000

many people started from the mathematical definitions of chaos for dynamical systems and adapted them to cellular automata. By using the usual product topology on $\{0,1\}^{\mathbb{Z}}$, the shift is necessarily chaotic, and for many possible applications of CAs, like roadways traffic for example, we see that these definitions are maladjusted. That is why Formenti introduced Besicovitch topology [4,2]. For this topology, the phase space is not locally compact, thus all the mathematical results become wrong or at least have to be proved again. On the other hand, starting from the physicist approach of chaos, that is high sensitivity to initial conditions, Bagnoli et al. [1] propose to measure chaos experimentally through Lyapunov exponents, that is, roughly, to evaluate the damage spreading speed when a single cell is modified.

In this article, we will formalize both approaches to study their relationships. First, we will define the almost everywhere sensitivity to initial conditions for Besicovitch topology and then we will partition the CAs whether the default number in average tends to zero, is bounded or not. We will also be interested in the definitions of Bμ-attracting sets and Dμ-attracting sets [10]. Let us notice that all the definitions depend on a measure.

We will begin by the definitions of cellular automata, Besicovitch topology and Bernoulli measure. In a second section, we will formalize damage spreading, almost everywhere sensitivity and μ-attracting sets. We will then study the relations that exist between the classes we defined. The last section speaks about the links with Wolfram's classification.

The extended version with the proofs is to be found as research report available by FTP [11].

1 Definitions

1.1 Cellular Automata

For simplicity, we will only consider unidimensional CAs in this paper. However, all the concepts we introduce are topological and it seems that there is no problem to extend them to higher dimensional CAs.

Definition 1. *A* **radius-r unidimensional cellular automaton** *is a couple* (Q,δ) *where* Q *is a finite set of states and* $\delta : Q^{2r+1} \longrightarrow Q$ *is a transition function. A configuration* $c \in Q^{\mathbb{Z}}$ *of* (Q,δ) *is a function from* $\mathbb{Z}$ *into* Q *and its global transition function* $G_\delta : Q^{\mathbb{Z}} \longrightarrow Q^{\mathbb{Z}}$ *is such that* $(G_\delta(c))(i) = \delta(c(i-r),...,c(i),...,c(i+r))$.

Notation 1 *Let us define*

$$\delta^{m+2r\to m} \left| \begin{array}{l} Q^{m+2r} \longrightarrow Q^m \\ x \longmapsto y \end{array} \right.$$

such that for all $i \in \{1,...,m\}$, $y_i = \delta(x_{i-r},...,x_i,...,x_{i+r})$.

Definition 2. *An* **Elementary Cellular Automaton (ECA)** *is a radius-1 two states (usually* 0 *and* 1*) unidimensional cellular automaton.*

For ECAs, we will use Wolfram's notation: they are represented by an integer between 0 and 255 such that the transition function of the CA number i whose writing in base 2 is $i = \overline{a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0}^2$ satisfies:

$$\begin{array}{ll} \delta_i(0,0,0) = a_0 & \delta_i(1,0,0) = a_4 \\ \delta_i(0,0,1) = a_1 & \delta_i(1,0,1) = a_5 \\ \delta_i(0,1,0) = a_2 & \delta_i(1,1,0) = a_6 \\ \delta_i(0,1,1) = a_3 & \delta_i(1,1,1) = a_7 \end{array}$$

Let us remark that CAs with different numbers may have the same behavior by switching the states 0 and 1, for instance $184 = \overline{10111000}^2$ and $226 = \overline{11100010}^2$. If r is a rule number, we will denote $\overline{r}$ the rule after exchanging the states and $\overleftrightarrow{r}$ the rule which has a symmetric behavior (see [5] for more details).

We will speak about the cellular automaton $120 = (\{0,1\}, \delta_{120})$ or equivalently of the rule 120.

In the general definition of additive CAs due to Wolfram, an additive CA is a CA that satisfies the superposition principle ($\delta(x + x', y + y', z + z') = \delta(x, y, z) + \delta(x', y', z')$). These CAs are very interesting to provide examples because their behavior obey algebraic rules adapted to a formal study while their space-time diagrams appear complicated. We will use here, like in [13,10], a more restrictive definition:

Definition 3. *We will call additive CA a unidimensional CA whose state set is* $\mathbb{Z}/n\mathbb{Z}$ *and whose transition function is of the form:*

$$\delta(x_{-1}, x_0, x_1) = x_0 + x_1 (mod\ n)$$

1.2 Besicovitch Topology

The most natural topology on CA configuration sets is the product topology. The problem is that this topology emphasizes what is happening closed to the origin while in many applications of CAs all the cells have the same importance. Thus, the adaptation of the mathematical notions of chaos to CAs for the product topology are not adapted: the shift is necessarily chaotic, that is not adapted to car traffic simulation for example. To propose more satisfying definitions of chaos, Formenti introduced Besicovitch pseudo-distance, that induce a shift invariant topology on the quotiented space:

Definition 4. *The Besicovitch pseudo-metric on* $Q^{\mathbb{Z}}$ *is given by*

$$d(c, c') = \limsup_{l \longrightarrow +\infty} \frac{\#\{i \in [-l, l] | x_i \neq y_i\}}{2l + 1} \qquad \textit{where } \# \textit{denotes the cardinality.}$$

Property 1. $Q^{\mathbb{Z}}$ quotiented by the relation $x \sim y \iff d(x,y) = 0$ with Besicovitch topology is metric, path-wise connected, infinite dimensional, complete, neither separable nor locally compact [2]. Furthermore, $x \sim y \Longrightarrow G_\delta(x) \sim G_\delta(y)$ and the transition function of a CA is a continuous map from $Q^{\mathbb{Z}}/\sim$ into itself.

Remark 1. Actually, the results of this paper are not specific to Besicovitch topology, but are true for a wide class of topologies including Weil one. A general study of Besicovitch like topologies has been done in [9] and an interesting question would be to determine those that behave like Besicovitch one and what happens for the other ones. The only reason we point this out here is that there are many ways to extend Besicovitch in higher dimensional grids: the extention of Besicovitch pseudo-metric on $Q^{\mathbb{Z}^n}$ is

$$d(c,c') = \limsup_{l \longrightarrow +\infty} \frac{\#\{i \in B(0,l) \subset \mathbb{Z}^n | x_i \neq y_i\}}{\#B(0,l)}$$

where $B(0,l)$ is a ball centered at the origin and of radius l in $\mathbb{Z}^n$ for an arbitrary chosen distance on $\mathbb{Z}^n$, for instance $\overline{d}_1(a_1,...,a_n) = |a_1| + ... + |a_n|$ or $\overline{d}_2(a_1,...,a_n) = (a_1^2 + ... + a_n^2)^{1/2}$. Of course different distances give different topologies but all of them are equivalent for our purpose because they differ on a null measure set. This is the only difficulty to extend the unidimensional concept to higher dimensional CAs because the definitions of measures and ergodicity given in the following section exist for any dimensional space.

1.3 Measure on the Configuration Set

Notation 2 *Let Q be a finite alphabet with at least two letters. $Q^+ = \cup_{n \geq 1} Q^n$ is the set of finite words on Q. The i^{th} coordinate $x(i)$ of a point $x \in Q^{\mathbb{Z}}$ will also be denoted x_i and $x_{[j,k]} = x_j ... x_k \in Q^{k+1-j}$ is the segment of x between indices j and k. The cylinder of $u \in Q^p$ at position $k \in \mathbb{Z}$ is the set*

$$[u]_k = \{x \in Q^{\mathbb{Z}} | x_{[k,k+p-1]} = u\}.$$

Let σ be the shift toward the left: $\sigma(c)_i = c_{i+1}$ (i.e. the rule number 85).

A Borel probability measure is a nonnegative function μ defined on Borel sets. It is given by its values on cylinders, satisfies $\mu(Q^{\mathbb{Z}}) = 1$, and for every $u \in Q^+$, $k \in \mathbb{Z}$,

$$\sum_{q \in Q} \mu[uq]_k = \mu[u]_k \text{ and } \sum_{q \in Q} \mu[qu]_k = \mu[u]_{k+1}$$

Definition 5. *A measure μ is σ-invariant if $\mu[u]_k$ does not depend on k (and will thus be denoted $\mu[u]$).*

Definition 6. *A σ-invariant measure is σ-ergodic if for every invariant measurable set Y $(\sigma(Y) = Y)$, either $\mu(Y) = 0$ or $\mu(Y) = 1$.*

Bernoulli measures are the most simple, this is the reason why we will use them in all our examples while the definitions and probably most of the theorems remain true for other σ-ergodic measures, for instance Markov measures (with correlations over a finite number of cells) or measures such that the correlation between two states decreases exponentially with their distance. Obviously, to study specific rules, like number preserving rules, Bernoulli measures are less interesting and we may want to consider other σ-ergodic measures. But, practically, these other measures will often be Bernoulli measures after a grouping operation.

Definition 7. *A Bernoulli measure is defined by a strictly positive probability vector $(p_q)_{q\in Q}$ with $\sum_{q\in Q} p_q = 1$ and if $u = u_0...u_{n-1} \in Q^n$, $\mu[u_0...u_{n-1}] = p_{u_0}...p_{u_{n-1}}$.*

We will use the following classical result: *the Bernoulli measures are σ-ergodic.*

For 2-states CAs, the Bernoulli measures will be denoted μ_ρ where $\rho = p_1 = 1 - p_0$ is the probability for a state to be 1.

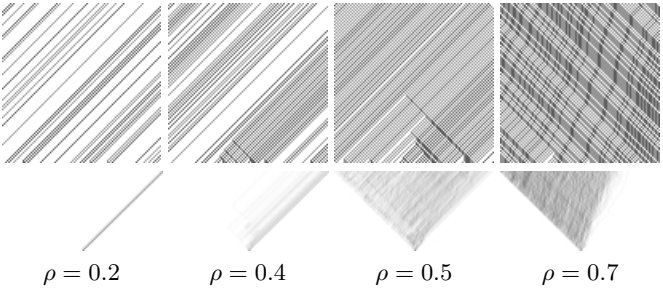


Fig. 1. The CA T is a very simple traffic model based on the rule 184 but with two different models of cars. The system seems "chaotic" when the density ρ of cars is greater than or equal to 0.5 because of the traffic jams, but not "chaotic" else. Below the space-time diagrams (time goes toward the top), we see with a grey level the space-time repartition of the average number of alterations induced by the modification of the middle cell.

On the figure 1, we see a very simple example of CA that changes of behavior depending on the density of cars on the railway. Saying that this CA is chaotic or not does not make sense since it will depend on its utilization: whether it is used for traffic jam or for fluid traffic simulation. Its average behavior makes no sense since we do not explain what is a random configuration, that is which measure we take on its configuration set. If we assume that the cars repartition is initially uniform and that we have the same number of red and blue cars, we will consider the Bernoulli measures μ^*_ρ such that the probability to find a blue car in

a cell is $\rho/2$ and equal to the probability to find a red car while the probability that there is no car is $1-\rho$. Now, it is possible to say (see below) that this CA is μ_ρ^*-almost everywhere sensitive to initial conditions when $\rho \geq 1/2$ while it is μ_ρ^*-almost never sensitive to initial conditions else. If it is important to take into account the fact that a lot of people take their cars at the same time to go to work, other measures allow to modelize a non uniform repartition.

2 Some Classification Tools on CAs

2.1 Damage Spreading

Inspired by Lyapunov exponents [1], we will define the damage spreading of a CA via a measure. The main difference is that we count the "effective" damages induced by a single cell modification, for instance, if the cell modification leads to two alterations after $t-1$ steps, that each of these alterations would change one state at time t but the action of both leads this state to remain the same, then rather than counting 2 modifications like in the Lyapunov exponents, we count 0 modification because the state did not change. It appears that the rule 210 (see figure 3) has a Lyapunov exponent higher than 1 (thus the number of modifications is exponential if we may count a cell many times) while its damage spreading (the average number of different cells) is bounded. Let us now define this formally:

Definition 8. *Let μ be a σ-ergodic measure on the set of configurations, let $\mathcal{A}$ be a CA and $\mathcal{C}_\mathcal{A}$ its configuration set. If $c \in \mathcal{C}_\mathcal{A}$, we will define*

$$c_{p \leftarrow s} \left| \begin{array}{l} \mathbb{Z} \longrightarrow Q_\mathcal{A} \\ x \longmapsto \begin{cases} c(x) \text{ if } x \neq p \\ s \text{ else} \end{cases} \end{array} \right.$$

that is the configuration whose state of the cell p is changed to s. Let us now define the dependence coefficients $\alpha_{\mu,\mathcal{A},t,p}$ which indicate the probability (according to μ) that the state of the cell 0 after t computation steps changes when we change the state at position p to q with probability π_q. Formally:

$$\alpha_{\mu,\mathcal{A},t,p} = \sum_{q \in Q_\mathcal{A}} \pi_q \mu(\{c \in \mathcal{C}_\mathcal{A} \mid (G^t_{\delta_\mathcal{A}}(c))_0 \neq (G^t_{\delta_\mathcal{A}}(c_{p \leftarrow q}))_0)$$

Remark 2.
- When no confusion about the measure and the cellular automaton is possible, we will simply denote the dependence coefficients $\alpha_{t,p}$.
- If r is the radius of the CA, $p > r \times t \implies \alpha_{t,p} = 0$ because the cell 0 state after t computation steps is independent of the cell p of the initial configuration.

Definition 9. *The damage spreading of a cellular automaton $\mathcal{A}$ according to a measure μ is the infinite sequence of positive real number*

$$\left(\sum_{p\in\mathbb{Z}} \alpha_{\mu,\mathcal{A},t,p}\right)_{t\in\mathbb{N}}$$

This sequence is well defined for all t thanks to the previous remark: the $\alpha_{\mu,\mathcal{A},t,p}$ are almost all equal to zero.

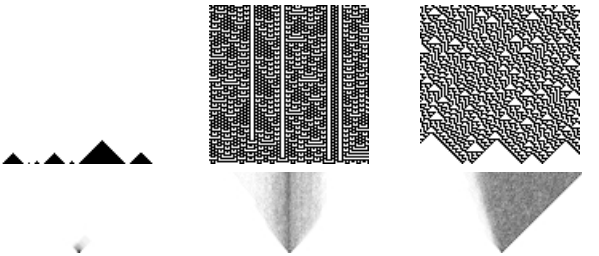


Fig. 2. The rule 128 belongs to $[\mathcal{D} \overset{\mu}{\to}^{+} 0]$, 109 to $[\mathcal{D} \overset{\mu}{\to}^{+} a]$ and 30 to $[\mathcal{D} \overset{\mu}{\to}^{+} +\infty]$. The bottom diagrams represent with grey level the probability of each cell to be affected by the modification of the middle cell.

This notion allows to define the class $[\mathcal{D} \overset{\mu}{\to}^{+} 0]$ of CAs whose damage spreading tends to zero, the class $[\mathcal{D} \overset{\mu}{\to}^{+} +\infty]$ of CAs whose damage spreading is not bounded (its limit sup tends to $+\infty$) and the class $[\mathcal{D} \overset{\mu}{\to}^{+} a]$ when the damage spreading limit sup tends to a finite non zero value. The figure 2 shows an example in each class. Obviously, these 3 classes define a partition of the set of CAs.

Theorem 1. *The additive CAs have non bounded damage spreading, they are in $[\mathcal{D} \overset{\mu}{\to}^{+} +\infty]$ (see [11] for the proof).*

2.2 μ-Almost Everywhere Sensitivity to Initial Conditions

Let us recall the classical definition of sensitivity to initial conditions:

Definition 10. *A CA is sensitive to initial conditions for a pseudo-distance d if there exists a constant $M > 0$ such that for all $\epsilon > 0$ and for all configurations c, there exists a configuration c' with $d(c, c') < \epsilon$ and an integer n such that $d(G^n_{\delta_{\mathcal{A}}}(c), G^n_{\delta_{\mathcal{A}}}(c')) \geq M$.*

The main reason of the introduction of μ-almost everywhere sensitivity is the study of some particular cases. On the one hand, the rule 120 appears sensitive to initial conditions but it seems that there exist some very artificial configurations that stop all the information transfer so that actually the rule 120 is not sensitive. On the other hand, the rule 210 does not appear to be sensitive, but is not equicontinuous (i.e. is sensitive to initial conditions on a subset of its configuration set) because of the configurations 0^* (see figure 3). Thus, in the same class, the elements of which are neither sensitive nor equicontinuous, we have two rules with very different behaviors. The idea is to say that 120 is almost everywhere sensitive, while 210 is almost never sensitive.

To define the almost everywhere sensitivity to initial conditions, we could just replace "for all configurations c" by "for μ-almost all configurations c" in the sensitivity definition. We will give a more restrictive (because of the first point of the next remark) definition so that a CA that is not μ-almost everywhere sensitive, is "μ-almost never" sensitive to initial conditions (see the third point of the next remark). Furthermore, because of the kind of proof we want to do, it is not more difficult to prove the μ-almost everywhere sensitivity for this definition.

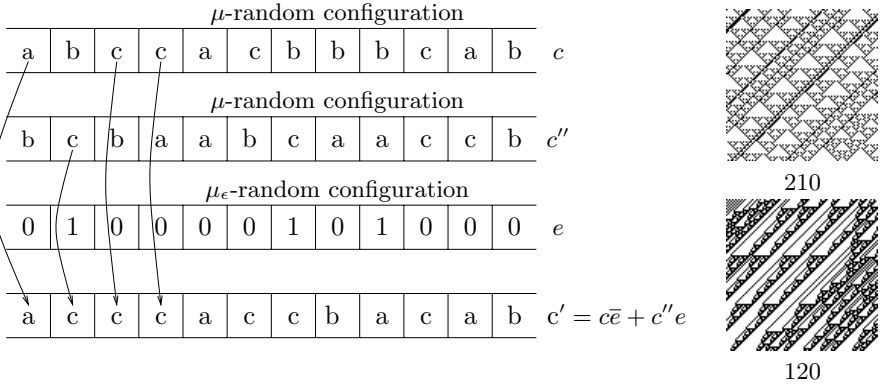


Fig. 3. The configuration $c' = c\overline{e} + c''e$ is the configuration whose state at a given position is equal to the corresponding state of c when the corresponding state of e is equal to 0 and to the corresponding state of c'' else. Let us remark that, due to the great number law, with probability 1, $d(c, c') = \epsilon d(c, c'') \leq \epsilon$.

Definition 11. *A CA is μ-almost everywhere sensitive to initial conditions (for Besicovitch pseudo-distance) if there exists $M > 0$ such that for all $\epsilon_0 > 0$, there exists $\epsilon < \epsilon_0$ such that if c and c'' are two μ-random configurations, if e is a μ_ϵ random configuration and if $c' = c\overline{e} + c''e$ is the configuration whose state at a given position is equal to the corresponding state of c when the corresponding state of e is equal to 1 and to the corresponding state of c'' else (see figure 3), then with probability 1 (for $\mu \times \mu \times \mu_\epsilon$) there exists n such that $d(G^n_{\delta_\mathcal{A}}(c), G^n_{\delta_\mathcal{A}}(c')) \geq M$.*

Remark 3. – This definition implies that there exists M such that for μ-almost all configurations c and for all $\epsilon > 0$, there exist c' and n with $d(c, c') < \epsilon$ and $d(G^n_{\delta_\mathcal{A}}(c), G^n_{\delta_\mathcal{A}}(c')) \geq M$;
– With the product topology, the previous result would imply the sensitivity to initial conditions, the point is that Bernoulli measures are of full support (i.e. the open sets have a non null measure), but it is not the case on $Q^{\mathbb{Z}}/\sim$ with Besicovitch topology;
– The set of configurations 3-uplets (c, c'', e) such that if $c' = c\overline{e} + c''e$ there exists n so that $d(G^n_{\delta_\mathcal{A}}(c), G^n_{\delta_\mathcal{A}}(c')) \geq M$ is obviously shift invariant on $(Q \times Q \times \{0,1\})^{\mathbb{Z}}$. As $\mu \times \mu \times \mu_\epsilon$ is σ-ergodic, thus the set measure is either 1 or 0. So a CA is either μ-almost everywhere sensitive to initial conditions or "μ-almost never sensitive to initial conditions": for any η there exists ϵ such that if we build c, c' as usual, for any n, $d(G^n_{\delta_\mathcal{A}}(c), G^n_{\delta_\mathcal{A}}(c')) \leq \eta$ $\mu \times \mu \times \mu_\epsilon$-almost everywhere.

The μ-almost everywhere sensitivity to initial conditions makes sense because we saw that some CAs are not (obviously the rule 0 is not) and we will prove that the additive CAs are:

Theorem 2. *The additive CAs are μ-almost everywhere sensitive to initial conditions for any non trivial Bernoulli measure μ (see [11] for the proof).*

μ-attracting sets μ-attracting sets have been defined in [10]. In this article, P. Kůrka and A. Maass study the links between attracting and μ-attracting sets for different topologies.

Definition 12. *A sub-shift is any subset $\Sigma \subseteq Q^{\mathbb{Z}}$, which is σ-invariant and closed in the product topology. The language $L(\Sigma)$ of a sub-shift $\Sigma \subseteq Q^{\mathbb{Z}}$, is the set of factors of Σ. A sub-shift is of finite type (SFT), if there exists a positive integer p called order, such that for all $c \in Q^{\mathbb{Z}}$,*

$$c \in \Sigma \iff \forall i \in \mathbb{Z}, c_{[i,i+p-1]} \in L(\Sigma).$$

Definition 13. *For a SFT $\Sigma \subseteq Q^{\mathbb{Z}}$ of order p and $x \in Q^{\mathbb{Z}}$, define the density of Σ-defects in x by*

$$d_D(x, \Sigma) = \limsup_{l \longrightarrow +\infty} \frac{\#\{i \in [-l, l] \mid x_{[i,i+k-1]} \notin L(\Sigma)\}}{2l+1}.$$

Notation 3 *When d is a pseudo-distance, we can naturally define the pseudo-distance from an element x to a set as the inf of the pseudo-distance between x and the elements of the set:*

$$d(x, \Sigma) = \inf_{y \in \Sigma} d(x, y).$$

Let us notice that $d_D(x, \Sigma)$ is not associated to any pseudo-distance because generally $d_D(x, \{y\}) \neq d_D(y, \{x\})$.

Definition 14. *Let μ be σ-ergodic measure, a sub-shift Σ is Bμ-attracting (resp. Dμ-attracting) if*

$$\mu(\{c \in Q^{\mathbb{Z}} \mid \lim_{n \to +\infty} \mathbf{d}(G_\delta^n(c), \Sigma) = 0\}) = 1$$

when $\mathbf{d} = d(= d_B), d_D$ respectively.

Remark 4.
- As the set $\{c \in Q^{\mathbb{Z}} \mid lim_{n \to +\infty} \mathbf{d}(f^n(c), \Sigma) = 0\}$ is shift invariant, its measure is either 0 or 1.
- From a Bμ-attracting sub-shift, we can easily extract a minimal Bμ-attracting sub-shift by considering the configurations c' of Σ such that $\mu(\{c \in Q^{\mathbb{Z}} \mid \liminf d_B(f^n(c), c') = 0\}) = 1$. And when μ is a Bernoulli measure, this implies that c' is uniform (i.e. of the form q^* for $q \in Q$).
- Furthermore, if a sub-shift is Bμ-attracting, then it is Dμ-attracting.

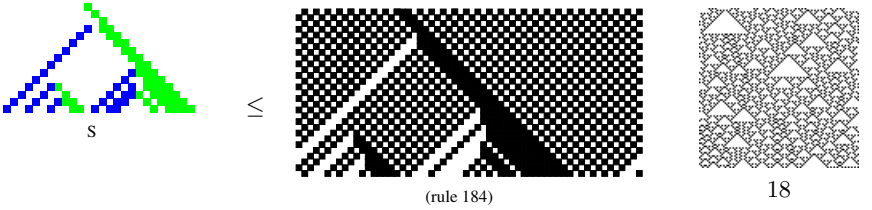


Fig. 4. $\{B^*\}$ is a Bμ-attracting (thus Dμ-attracting) set of the CA $\mathcal{S}$ which has 3 states, one is going to the right, one is going to the left (the third one is B, the blank one, into which the other state may move) and when they meet, both are annihilated. When μ is a measure such that the number of states going to the left and to the right have the same probability of presence on the initial configuration, then the uniform configuration composed by the blank state is a Bμ-attracting set. As $\mathcal{S}$ is a sub-automaton of 184^2 where 184^2 is the CA 184 whose states are grouped two by two (see [12]), we have the same result on 184^2 for the measure so that $(1,0)$ has probability 0 and the probability of $(1,1)$ and $(0,0)$ are equal. This measure on $(\{0,1\} \times \{0,1\})^{\mathbb{Z}}$ corresponds to a non shift invariant measure on $\{0,1\}^{\mathbb{Z}}$, and actually, 184 has no Bμ-attracting set when μ is a non trivial shift-invariant measure. The point is that for $\mu_{1/2}$ (so that particles going to the left and to the right have the same probability), the sub-shift $\{(01)^*, (10)^*\}$ is Dμ-attracting, but asymptotically, the configurations tend to be at pseudo-distance $1/2$ of both configurations. The rule 18 seems to be another example of CA with Dμ-attracting sets, but no Bμ-attracting set.

The definition of Bμ-attracting sets is very natural: a set is Bμ-attracting if from almost all configurations (w.r.t. μ), the (Besicovitch) pseudo-distance between the successive configurations and the sub-shift tends to 0. We saw in the remark that in this case, almost all evolutions tend to uniform configurations when μ is a Bernoulli measure. The rule 128 (see figure 2) or the CA $\mathcal{S}$

and 184^2 for some measure μ (see figure 4) have a Bμ-attracting set. The definition of Dμ-attracting sets is more topological because it only depends on the language $L(\Sigma)$ and does not take care of how many times a pattern (a word of $L(\Sigma)$) appears. Thus, Dμ-attracting sets are more powerful and allow to point out homogenization process to periodical configurations and not only to uniform configurations. For instance, as proved in [10], the sub-shift $\{(01)^*, (10)^*\}$ is B$\mu_{1/2}$-attracting for the rule 184. As noticed in [10], 18 seems to be an example of CA with a Dμ-attracting set which is not of finite type: $\{c \in \{0,1\}^{\mathbb{Z}} | \forall i, c(2i) = 0\} \cup \{c \in \{0,1\}^{\mathbb{Z}} | \forall i, c(2i+1) = 0\}$.

In the following, we will only use Dμ-attracting sets to point out homogenization to periodical configurations. In this case, we see that the whole information of the initial configuration is erased, formally, the metric entropy of the successive configurations tends to zero.

Definition 15. *Let $(\mathcal{A}, \delta)$ be a CA and μ a σ-ergodic measure, the metric entropy of its configuration after t computation steps is defined as follow:*

$$S_\mu^{(t)}(\mathcal{A}) = \lim_{n \to +\infty} \frac{-\sum_{u \in Q^n} p_u^{(t)} ln(p_u^{(t)})}{n}$$

with the usual convention $0 \times log(0) = 0$ and where p_u is the probability of apparition of the pattern u in the configuration c:

$$p_u^{(t)} = G_\delta^t(\mu)([u]_0)$$

where the notation $f(\mu)$ represents, as usual, the measure defined by $f(\mu)(X) = \mu(f^{-1}(X))$. In mathematical terminology, $S_\mu^{(t)}(\mathcal{A})$ is the metric entropy of σ for the measure $G_\delta^t(\mu)$.

Definition 16. *The class of CAs that erase all their initial configuration information will be denoted $[S_\mu \to 0]$ and formally defined as follows: a CA is in $[S_\mu \to 0]$ if and only if*

$$S_\mu^{(t)}(\mathcal{A}) \longrightarrow_{t \to +\infty} 0.$$

Theorem 3. *If a CA has a Dμ-attracting set of null topological entropy, then it is in $[S_\mu \to 0]$ (see [11] for the proof).*

Definition 17. *The topological entropy of a sub-shift Σ is*

$$S_\Sigma(\sigma) = \lim_{n \longrightarrow +\infty} \frac{ln(\#\{u \in Q^n \mid \exists c \in \Sigma, c_{[0,|u|]} = u\})}{n}.$$

3 Relations between Damage Spreading, μ-Ae Sensitiveness and the Existence of a Bμ-Attracting Set

In this section, we always assume that μ is a Bernoulli measure.

Theorem 4. *If $\mathcal{A}$ μ-damage spreading tends to 0 then there exists a set of uniform configurations which is Bμ-attracting (see [11] for the proof).*

Reciprocally, there are CAs with Bμ-attracting sets that are not in $[\mathcal{D} \overset{\mu}{\to}^{+} 0]$: the rule $\mathcal{S}$ (see figure 4) with 3 states (a blank one B into which one state l goes to the left and a state r goes to the right, the collision of two particles leads to their annihilation) is in $[\mathcal{D} \overset{\mu}{\to}^{+} a]$. In addition, we will describe later a CA that experimentally seems to be in $[\mathcal{D} \overset{\mu}{\to}^{+} +\infty]$ but tends to a uniform configuration.

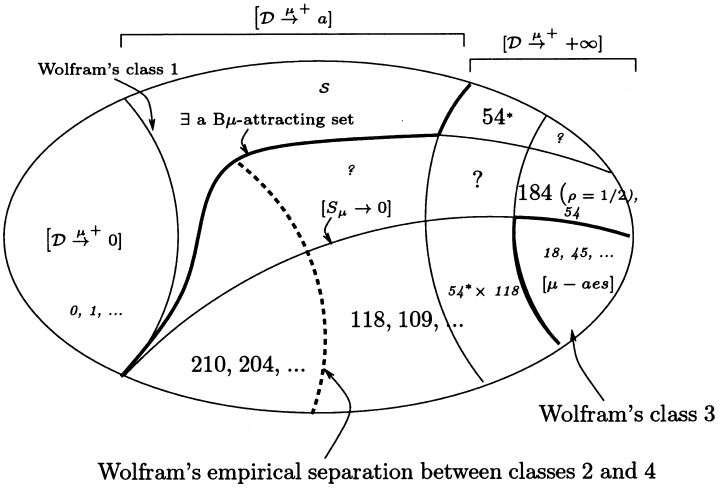


Fig. 5. Relations between damage spreading, μ-ae sensitiveness and the existence of Bμ-attracting sets when μ is a Bernoulli measure

Let us now investigate the relations between damage spreading and μ-almost everywhere sensitiveness. The idea is to take a μ-random configuration c. We then build c' from c: for all the cells, the state of c' is equal to the state of c with probability $1 - \eta$, is equal to q with probability ηp_q. With probability 1, the Besicovitch pseudo-distance $d(c, c')$ between c and c' is $\eta d(c, c'') \leq \eta$. So we can prove the following theorem.

Theorem 5. *Being μ-almost everywhere sensitive to initial conditions implies to have non bounded damage spreading (i.e. $[\mu - aes] \subseteq [\mathcal{D} \overset{\mu}{\to}^{+} +\infty]$) (see [11] for the proof).*

It is experimentally easy to observe but it seems difficult to prove that 184 is $\mu_{1/2}$-almost everywhere sensitive to initial conditions. To understand what

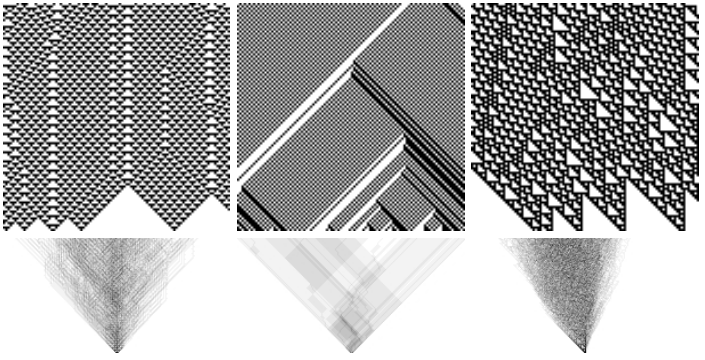


Fig. 6. 54 and 184 for $\mu_{1/2}$ are in $[\mathcal{D} \overset{\mu}{\to}^{+} +\infty] \cap [S_\mu \to 0]$, we do not know where is 110

happens, let us consider a random walker on $\mathbb{Z}$ starting at 0 and reading the initial configuration from the cell 0 toward the right (resp. left). When he reads 1 he goes toward the right and when he reads 0, toward the left. If we change the cell 0 to 1 (resp. to 0), it generates at least one modification on all the configurations. This modification moves to the right when the random walker is on $\mathbb{Z}_+^*$, to the left when it is in $\mathbb{Z}_-^*$ and do not move if the random walker is on 0. Sometimes, because of this modification, a whole region of background shifts and all the cells of this region are changed. Actually, the overall evolution of 184 leads to bigger and bigger homogeneous regions of 01^* and 10^*, but when we take two generic initial configurations, there is no reason that these regions match, thus, asymptotically, the pseudo-distance between the configurations after many computation steps is 1/2.

It seems that the rule 54 also belongs to $[\mu - aes] \cap [S_\mu \to 0]$. Actually, g_e particles disappearance is an irreversible phenomenon as proved in [12] that will occur more and more rarely when the particles g_e become fewer and fewer but the number of g_e particles tends to 0 as confirmed by [3,8] experiments. Then if any interaction between w and g_0 particles occurs, one particle disappears, we can think that the sub-shift $\{0001^*, 1110^*\}$ is $\mathrm{D}\mu_\rho$-attracting for $0 < \rho < 1$.

The previous theorem raises a natural question: are the CAs with unbounded damage spreading μ-almost everywhere sensitive to initial conditions?

It seems that no. Let us consider a CA 54^* that simulates the rule 54 but in a uniform background. Such a CA can be formally defined thanks to Hanson and Crutchfield's filter [8] which is a CA but only on "valid" configurations. Here, we consider any extension of this CA for a measure such that a generic configuration is correct with probability 1. Experimentally, the particle number decreases like $t^{-1/2}$ and thus tends to 0 so that this CA for well chosen measures tends to a uniform configuration. Due to the slow particle number decreasing, this CA seems to have non bounded damage growth: with a non null probability, a single perturbation creates or suppresses one or more particles. At each interaction of

a defect particle with a particle, the defect is duplicated, the probability of such collisions is linked to the density of particles, thus the average number of defects should look like $\int x^{-1/2} = 2\sqrt{x}$ which tends to $+\infty$ when $x \longrightarrow +\infty$, this CA seems to have non bounded damage growth (that is experimentally observed). Furthermore this CA does not seem to be almost everywhere sensitive to initial conditions because of the following theorem:

Theorem 6. *Let A be a CA that almost everywhere tends to a uniform configuration, A is not almost everywhere sensitive to initial conditions (see [11] for the proof).*

The question to know whether there are almost everywhere sensitive CAs with a Bμ-attracting set is open.

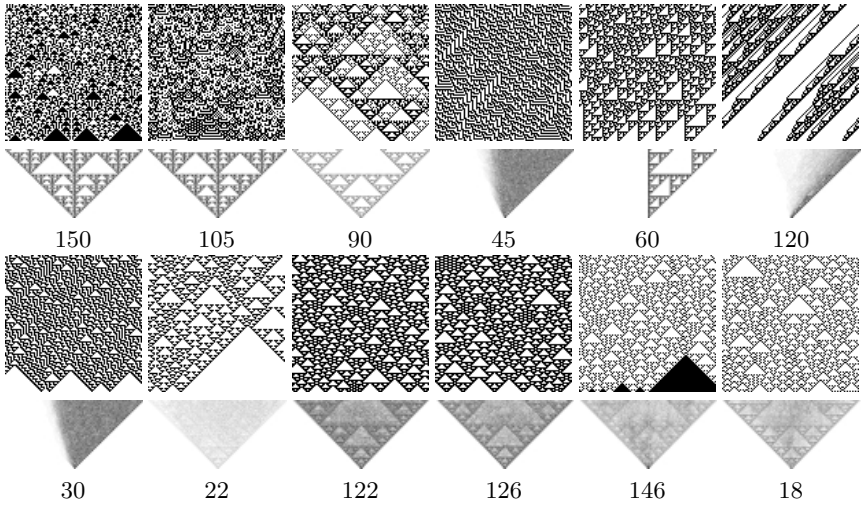


Fig. 7. All the chaotic behaviors among 2 states unidimensional CAs

4 Links with Wolfram's Classification

The CAs in $[\mu - aes] \backslash [S_\mu \to 0]$ practically match on ECAs with the CAs that Wolfram put in his class 3, they are represented on the figure 7. It is not sure that this remains true for more complicated (with more states, in higher dimension or with a bigger neighborhood) CAs because they may present a lot of different behaviors depending on the initial configuration. Anyway, if we assume that this is a good formalization, Wolfram's observation that "the value of a particular site depends on an ever-increasing number of initial site values" is proved. Furthermore, we know that this condition is not sufficient to imply chaoticity.

Wolfram's class 4 on ECAs seems to split into two parts, the class $[\mu - aes] \cap [S_\mu \to 0]$ and a part of the class $[\mathcal{D} \overset{\mu}{\to}^+ a] \backslash [S_\mu \to 0]$. But the definition of this class (existence of particles) or the conjecture of the universality of its elements let us think that this class is completely independent of our criteria. The main point is that it is easy to build universal CAs in any non empty class since there are some in the class $[\mathcal{D} \overset{\mu}{\to}^+ 0]$ (a CA is usually universal on a null measure set).

The evolution of CAs that have Bμ-attracting sets tends μ_ρ-almost everywhere to a set of uniform configurations. Their behaviors look like the behaviors of Wolfram's class 1 CAs that "evolve after a finite number of time steps from almost all initial states to a unique homogeneous state, in which all sites have the same value". Actually, it seems natural to think that the rule 128 (see figure 2) which erases a succession of n states 1 in $n/2$ time steps is in the class 1 and this is confirmed by the examples of class 1 CAs given by Wolfram. But, if 0^* is obviously a Bμ-attracting set, with probability 1, the evolution does not converge to 0^* after a finite number of time. The point is that the probability to find a sequence of n successive 1 on the configuration is 1 whatever n. Next, Wolfram writes that "their evolution completely destroys any information on the initial state". We proved this fact for CAs with Bμ-attracting sets, but we also saw that complicated rules like 184 for $\rho = 1/2$ completely destroy any information on the initial state.

The big problem would be to find a way to split the CAs whose damage spreading is bounded but does not tend to 0 in such a way that rules like 118 or 109 are not together with the identity. If the behavior of the damage growth of a CA is almost independent of the measure we take, we can separate the CAs whose damage spreading is uniformly bounded from the others. This would separate the identity from 118 and 109, but the rule 210 would be in the second case, that was not really expected.

Conclusion and Open Questions

One of the main conclusion of this article is that our intuitive property of chaos does not allow to split the set of CAs into two classes because some of them may have a chaotic or non chaotic behavior depending on the way to choose a random configuration. In addition, we see that Besicovitch topology that has been specifically introduced in CAs to express an intuitive notion of chaos is effectively a very interesting notion. Note that the introduction of a measure is very helpful to deal with the too wide class of neither sensitive nor equicontinuous CAs. In this article, we introduce a new Lyapunov like notion that allows to measure information diffusion. This notion appears more precise than the other ones but still not enough to ensure a chaotic behavior. Finally, the introduced notions allow to formalize some of Wolfram's observations and thus to prove how relevant they are.

A lot of open questions remains, among them

- to find examples in or to prove the emptiness of $[\mu - aes]$ with a Bμ-attracting set and of CAs in $[S_\mu \to 0]$ with no Bμ-attracting set and not in $[\mu - aes]$.
- the generalization for more general measures (with exponentially decreasing correlations) of the theorems.
- to find a "good" definition that separates the identity from 118 in $[\mathcal{D} \overset{\mu}{\to}^{+} a]$.

References

1. F. Bagnoli, R. Rechtman, and S. Ruffo. Lyapunov exponents for cellular automata. In M. L pez de Haro and C. Varea, editors, *Lectures on Thermodynamics ans Statistical Mechanics*. World Scientific, 1994.
2. F. Blanchard, E. Formenti, and P. Kůrka. Cellular automata in the Cantor, Besicovitch and Weil topological spaces. *Complex Systems*, to appear 1999.
3. N. Boccara, J. Nasser, and M. Roger. Particle-like structures and their interactions in spatio-temporal patterns generated by one-dimensional deterministic cellular-automata rules. *Physical Review A*, 44:866–875, 1991.
4. C. Cattaneo, E. Formenti, L. Margara, and J. Mazoyer. Shift invariant distance on $s^{\mathbb{Z}}$ with non trivial topology. In *Proceedings of MFCS' 97*. Springer Verlag, 1997.
5. G. Cattaneo, E. Formenti, L. Margara, and G. Mauri. Transformation of the one-dimensional cellular automata rule space. *Parallel Computing*, 23:1593–1611, 1997.
6. G. Cattaneo, E. Formenti, L. Margara, and G. Mauri. On the dynamical behavior of chaotic cellular automata. *Theoretical Computer Science*, 217:31–51, 1999.
7. R. H. Gilman. Classes of linear automata. *Ergod. Th. & Dynam. Sys.*, 7:105–118, 1987.
8. J. E. Hanson and J. P. Crutchfield. Computational mechanics of cellular automata: an example. *Physica D*, 103:169–189, 1997.
9. V. Kanovei and M. Reeken. On Ulam's problem of approximation of non-exact homomorphisms. *preprint*, 2000.
10. P. Kůrka and A. Maass. Stability of subshifts in cellular automata. *preprint*, 1999.
11. B. Martin. Damage spreading and μ-sensitivity on CA - extended to proofs. Research Report RR2000-04, LIP, ENS Lyon, 46 allée d'Italie - 69 364 Lyon Cedex 07, 2000. `ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2000/RR2000-04.ps.Z`.
12. B. Martin. A group interpretation of particles generated by one dimensional cellular automaton, 54 wolfram's rule. *Int. Journ. of Mod. Phys. C*, 11(1):101–123, 2000.
13. J. Mazoyer and I. Rapaport. Inducing an order on cellular automata by a grouping operation. In *STACS'98*, volume 1373, pages 116–127. Lecture Notes in Computer Science, 1998.
14. K. Čulik, J. Pach, and S. Yu. On the limit set of cellular automata. *SIAM Journal on Computing*, 18:831–842, 1989.
15. S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

Discrepancy Theory and Its Application to Finance

Shu Tezuka

IBM Tokyo Research Laboratory,
1623-14, Shimotsuruma, Yamato,
Kanagawa, Japan 242-8502,
tezuka@jp.ibm.com,
and
Research Center for Advanced Economic Engineering,
University of Tokyo,
4-6-1 Komaba, Tokyo, Japan 153-8904
tezuka@aee.u-tokyo.ac.jp

Abstract. In this paper, we first give a brief overview of discrepancy theory, then introduce low-discrepancy sequences, in particular, the original Faure and generalized Faure sequences. Next, we describe how to apply them to the problem of pricing financial derivatives, along with a successful application of this technique to the valuation of the present value of mortgage-backed securities (MBS). Finally, we will discuss future research directions.

1 Introduction

Discrepancy theory is a branch of Number Theory, whose historical origin is the theory of uniform distribution developed by H. Weyl and other mathematicians in the early days of the 20th century [1,2,6]. While the latter deals with the uniformity of infinite sequences of points, the former with the uniformity of finite sequences. Finite sequences always have some irregularity or deviation from the ideal uniformity due to their finiteness. *Discrepancy* is a mathematical notion for measuring such irregularity. For this reason, discrepancy theory is sometimes called as the theory of irregularities of distribution.

Quasi-Monte Carlo methods are one of the most successful applications of discrepancy theory. While Monte Carlo methods assume random numbers to provide probabilistic error bounds via the central limit theorem, quasi-Monte Carlo methods use *low-discrepancy sequences*[1] to allow deterministic error bounds via the Koksma-Hlawka theorem. The idea underlying low-discrepancy sequences is to use the point sets, not randomly distributed, but very uniformly distributed throughout the domain of integration. The extent to which the points are uniform has been mathematically defined as their discrepancy. The more uniformly distributed the points are, the lower the discrepancy is.

[1] Some people call them *quasi-random sequences*, but this term is a misnomer, since low-discrepancy sequences are totally deterministic.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 243–256, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Monte Carlo simulations are a common technique in finance, particularly in derivative pricing and in VaR (Value at Risk) computations, since they are simple to understand and to implement. In the area of derivative pricing, stochastic differential equations have been widely used as a modeling tool to describe the time evolution of the price of the underlying financial asset. The more complex stochastic models become, the less available analytical solutions are. Thus, Monte Carlo simulations are viewed as a last resort for the computation involved in these applications. On the other hand, the more accurate the solution is required to be, the more computing time is needed. But, Monte Carlo simulations are quite unsatisfactory due to their main drawback, that is to say, their notoriously slow convergence rate. According to the central limit theorem, the convergence rate is $O(1/\sqrt{N})$, where N is the number of sample points.

The use of low-discrepancy sequences for finance problems began around 1992, by Paskov and Traub [16]. They reported that quasi-Monte Carlo methods performed very well relative to simple Monte Carlo methods, as well as to antithetic Monte Carlo methods for pricing a ten-tranche CMO (Collateralized Mortgage Obligation), which they obtained from Goldman-Sachs[15]. Since then, many people (e.g., see the reference in [20]) have followed them and confirmed their finding with different pricing problems by using different types of low-discrepancy sequences.

The organization of this paper is as follows: Section 2 briefly overviews discrepancy theory: the original definition of discrepancy and its several variants. In Section 3, we introduce quasi-Monte Carlo methods, which is the deterministic version of Monte Carlo methods, and describe a general method of constructing low-discrepancy sequences, then in detail the so-called generalized Faure sequences. In Section 4, we give numerical experiments with the problem of computing the present values of MBS, along with some discussion of the results. In the last section, we conclude the paper with important research topics.

2 Brief Overview of Discrepancy Theory

2.1 Definition of Discrepancy

The formal definition of discrepancy is as follows: For N points $X_0, X_1, ..., X_{N-1}$ in $[0,1]^k$, we denote

$$\mathrm{I}(f) = \int_{[0,1]^k} f(x_1, ..., x_k) dx_1 \cdots dx_k,$$

and

$$\mathrm{Q}_N(f) = \frac{1}{N} \sum_{i=0}^{N-1} f(X_i). \tag{1}$$

We define a subinterval to be $J = [0, u_1) \times \cdots \times [0, u_k)$, where $0 < u_h \leq 1$ for $1 \leq h \leq k$, and also define the characteristic function as $\chi_J(x) = 1$ if

$x \in J$; otherwise 0. We commonly have the following two different definitions of discrepancy:

Definition 1. *The L_∞-discrepancy is defined by*

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,\infty} \stackrel{\mathrm{def}}{=} \sup_J | \mathrm{I}(\chi_J) - \mathrm{Q}_N(\chi_J) | ,$$

where the supremum is extended over all subintervals J.

Definition 2. *The L_2-discrepancy is defined by*

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,2} \stackrel{\mathrm{def}}{=} \left(\int_{[0,1]^k} (\mathrm{I}(\chi_J) - \mathrm{Q}_N(\chi_J))^2 \, du_1 \cdots du_k \right)^{1/2} . \tag{2}$$

By expanding the righthand-side, the L_2-discrepancy is explicitly written as

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,2}^2 = \int_{I^k} K(x,y) dx dy - \frac{2}{N} \sum_{i=1}^{M} \int_{I^k} K(x, x_i) dx + \frac{1}{N^2} \sum_{i,j=1}^{M} K(x_i, x_j),$$

where

$$K(s,t) = \int_{I^k} \chi_{[0,u)}(s) \chi_{[0,u)}(t) du = \prod_{i=1}^{k} (1 - \max(s_i, t_i)). \tag{3}$$

The well known lower and upper bounds for the discrepancy are as follows:
For any N point set, the L_2-discrepancy satisfies

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,2} = \Omega(\frac{(\log N)^{(k-1)/2}}{N}).$$

Roth (see, e.g., [1,6]) already proved the matching upper bound for this lower bound, although his proof was un-constructive[2]. Regarding the L_∞-discrepancy, we have the same lower bound as the L_2-discrepancy, but the upper bound is given as

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,\infty} = O(\frac{(\log N)^{k-1}}{N}).$$

It is conjectured [1] that this order of magnitude in the upper bound is optimal, i.e., that the lower bound can be further improved.

2.2 Several Variations of Discrepancy

In what follows, we give several variations of definition of discrepancy.

Basis functions:
From the definition in equation (2), the discrepancy can be seen as an integration

[2] Henri Faure informed me that Chen and Skriganov recently found a constructive proof.

error of the indicator function. So, by choosing "basis" functions different from the indicator function, we have different definitions of discrepancy. One example is due to Paskov [14], who chose

$$f_u^r(x) = \begin{cases} (u-x)^r/(r!)^k & \text{if } u > x \\ 0 & \text{otherwise} \end{cases}$$

as a basis function. Note that the case of $r = 0$ is equivalent to the original definition (2). In this case, $K(s,t)$ in (3) is changed to

$$K(s,t) = \int_{I^k} f_u^r(s) f_u^r(t) du.$$

If we regard the bivariate function $K(s,t)$ as a reproducing kernel on some Hilbert space, we can derive more general version of Koksma-Hlawka theorem, i.e., the worst case error estimate of integration. If we see $K(s,t)$ as a covariance kernel of some Gaussian measure, we can derive more general version of Woźniakowski theorem, the average case error estimate of integration. For more information, see Hickernell [3]. Both of Koksma-Hlawka and Woźniakowski theorems are mentioned in more detail in the next section.

Subspaces:
Another variant of the discrepancy is obtained by changing subintervals J (as well as I) to be another subspace such as half space, circle(or sphere), convex etc. In terms of the order of magnitude of the discrepancy, there are roughly two families. One is that the region J consists of scaled and translated copies of a fixed polygon or polytopes. The original definition is included in this family because J is a class of all axis-parallel boxes, in which no rotation is allowed. For this family, the discrepancy is bounded from above and from below by some constant powers of $\log N$ divided by N. For the other family, we can allow rotation beside scaling and translation. For example, arbitrarily rotated rectangular boxes are included in this family. Also, this family includes a class of convex bodies, for which Schmidt obtained a lower bound as $\Omega(N^{-2/(d+1)})$. And the upper bound is obtained by Beck ($d = 2$) as $N^{-2/3} \log^4 N$ and by Stute ($d \geq 3$) as $N^{-2/(d+1)} \log^c N$, where $c = 1.5$ for $d = 3$ and $c = 2/(d+1)$ for $d \geq 4$. For this family, the discrepancy is bounded by the order of some negative fractional power of N. See [1,6] for more details.

Nonuniform weights:
Another interesting generalization of the definition of discrepancy is the use of non-uniform (or unequal) weights. Then, the equation (1) is replaced by

$$\mathrm{Q}_{N,w}(f) = \sum_{i=0}^{N-1} w_i f(X_i),$$

where w_i are all real numbers. If $w_i = 1/N$ for all i, the original definition follows. This kind of definition is particularly useful for the error analysis in numerical

integration, where non-uniform weights are commonly used in quadrature (and cubature) formulas, e.g., Simpson rule, Gauss rule, etc. By exploiting this definition, Wasilkowski and Woźniakowski [22] obtained the following remarkable result:

Theorem 1. *There exists a cubature formula $Q_{N,w}$ such that*

$$\| \mathrm{I}(\chi_J) - \mathrm{Q}_{N,w}(\chi_J)\|_{k,2} \leq \epsilon$$

for $N(\epsilon, k)$ satisfying

$$N(\epsilon, k) \leq A\epsilon^{-1.4778..},$$

where A is an absolute constant.

If we compare the above result with the Monte Carlo methods, which need the number of sample points as

$$N(\epsilon, k) = O(\epsilon^{-2})$$

to achieve the standard deviation of ϵ, then its significance becomes clear. Another important point is that A is not depending on the dimension k. No similar results have been so far obtained for the discrapncy with uniform weights.

Combinatorial discrepancy:
The most abstract variant of the definition of the discrepancy is combinatorial discrepancy (or red-blue discrepancy): Let X be a set with cardinality n, and let S be a system of subsets of X. A mapping $\chi : X \to \{-1, 1\}$ is called a coloring. The discrepancy of S is defined as

$$disc(S) = \min_{\chi} disc(S, \chi)$$

where $disc(S, \chi) = \max_{A \in S} |\chi(A)|$ and $\chi(A) = \sum_{x \in A} \chi(x)$. The following upper bound can be derived by using a random coloring:

Theorem 2. *We have*

$$disc(S) = O(\sqrt{n \log m}),$$

where we assume A have at most m subsets.

Below, we give two typical examples of a set system (X, S).

Example 1. Consider a graph (V, E). Let V be a set X, and S be a set of subsets $A(v)$, where $A(v)$ denotes a set of neighboring nodes of $v \in V$. Then, the discrepancy with respect to this set system (V, S) is known as a very useful measure for the graph coloring problem, which play an important role in theoretical computer science.

Example 2. Consider the set $X = \{1, 2, ..., n\}$ and let S be a set of all subsets $\{a, a + b, a + 2b, ...\} \cap X$, where a and b are any integers. The discrepancy with respect to this set system (X, S) is known closely related to a famous Van der Waerden theorem in number theory.

More details and further discussions on discrepancy in general, see Matousek's excellent book[6].

3 Quasi-Monte Carlo Methods

3.1 Numerical Integration and Low-Discrepancy Sequences

Finance-related Monte Carlo problems, particularly those related to derivative pricing, can be formulated as problems of computing multidimensional integrals. The dimension of the integration is usually equal to the number of time steps by which the time interval under consideration is discretized. Once a problem can be formulated as one of numerical multidimensional integration, we have several "deterministic" cubature formulas. However, the direct extensions of one-dimensional quadrature formulas, such as trapezoidal rules, to higher-dimensional ones do not work well, because of the curse of dimensionality [21]. For example, the error bound for the k-dimensional trapezoidal rules is known to be $O(N^{-2/k})$, which means that the error grows exponentially as the dimension size becomes larger.

Low-discrepancy sequences have been considered as a promising alternative to high-dimensional numerical integration because its asymptotic error bound is $O((\log N)^k/N)$ for the k-dimensional integration problem. The Koksma-Hlawka theorem gives an important relation between discrepancy and numerical integration [8]:

Theorem 3 (Koksma-Hlawka). *If the integrand f is of bounded variation on the k-dimensional unit hypercube $[0,1]^k$ in the sense of Hardy and Krause, then for any $X_0, X_1, ..., X_{N-1} \in [0,1)^k$ we have*

$$|\ \mathrm{I}(f) - \ \mathrm{Q}_N(f)| \le \|\ \mathrm{I} - \ \mathrm{Q}_N\|_{k,\infty}\ \|f\|,$$

where $\|f\|$ is the Hardy-Krause variation of f.

What is most important here is that the bound is a product of two separate elements: one is dependent only on the point set and independent on f; the other only on the integrand and independent on the point set. For a given integrand f, if we can choose a point set so as to make the discrepancy as small as possible, we have the smallest integration error. We also have another important result, the Woźniakowski theorem [23]:

Theorem 4 (Woźniakowski). *Let C_k be the class of real continuous functions defined on $[0,1]^k$ equipped with the classical Wiener sheet measure w (that is, Gaussian with mean zero and covariance kernel*

$$R(\mathbf{s},\mathbf{t}) \stackrel{\text{def}}{=} \int_{C_k} f(\mathbf{s}) f(\mathbf{t})\ w(df) = \min(\mathbf{s},\mathbf{t}) \stackrel{\text{def}}{=} \prod_{j=1}^{k} \min(s_j, t_j)$$

for any vectors $\mathbf{s} = (s_1, ..., s_k)$ and $\mathbf{t} = (t_1, ..., t_k)$ in $[0,1]^k$). Then, for a given set of points $X_i = (x_{i1}, ..., x_{ik}), i = 0, 1, ..., N-1$, in $[0,1]^k$, we have

$$\int_{C_k} (\ \mathrm{I}(f) - \ \mathrm{Q}_N(f))^2\ w(df) = \|\ \mathrm{I} - \ \mathrm{Q}_N\|^2_{k,2,J},$$

where $\|\ \mathrm{I} - \ \mathrm{Q}_N\|_{k,2,J}$ is defined as the L_2-discrepancy with $J = (u_1, 1] \times \cdots \times (u_k, 1]$ in equation (2), where $0 \le u_h < 1$ for $1 \le h \le k$.

The theorem means that on average the integration error is dependent only on the discrepancy, not on the integrand f. Both of the above theorems tell us that the lower the discrepancy is, the smaller the integration error will be.

We now introduce the formal definition of low-discrepancy sequences:

Definition 3. *If a sequence $X_0, X_1, ...$ in $[0,1]^k$ satisfies the condition that for all $N > 1$, the discrepancy of the first N points is*

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,\infty} \leq C_k \frac{(\log N)^k}{N},$$

where C_k is a constant depending only on the dimension k, then we call it a low-discrepancy sequence.

Notice that the order of magnitude $(\log N)^k/N$ in the right-hand side is believed to be the optimal upper bound. Therefore, the sole difference among the many types of low-discrepancy sequences is how small the constant factor C_k is. In this article, we concentrate on the construction of low-discrepancy sequences based on (t,k)-*sequences in base* b. Before introducing them, we need the following definitions:

Definition 4. A b-ary box is an interval of the form

$$E = \prod_{h=1}^{k} [a_h b^{-d_h}, (a_h + 1) b^{-d_h})$$

with integers $d_h \geq 0$ and integers $0 \leq a_h < b^{d_h}$ for $1 \leq h \leq k$.

Definition 5. Let $0 \leq t \leq m$ be an integer. A (t, m, k)-net in base b is a point set of b^m points in $[0,1]^k$ such that every b-ary box of volume b^{t-m} contains exactly b^t points of the point set.

Now, we define (t,k)-sequences in base b.

Definition 6. Let $0 \leq t \leq m$ be an integer. A sequence of points $X_0, X_1, ...$, in $[0,1]^k$ is called a (t,k)-sequence if for all integers $j \geq 0$ and $m > t$, the point set consisting of $[X_n]_m$ with $jb^m \leq n < (j+1)b^m$ is a (t,m,k)-net in base b, where $[X]_m$ denotes the coordinate-wise m-digit truncation in base b of X.

Following Sobol' and Faure's results, Niederreiter [8] obtained the following theorem for an arbitrary integer base $b \geq 2$:

Theorem 5 (Niederreiter). *For any $N > 1$, the discrepancy of the first N points of a (t,k)-sequence in base b satisfies*

$$\| \mathrm{I} - \mathrm{Q}_N \|_{k,\infty} \leq c(t,k,b) \frac{(\log N)^k}{N} + O\left(\frac{(\log N)^{k-1}}{N} \right),$$

where $c(t,k,b) \approx \frac{b^t}{k!} (\frac{b}{2 \log b})^k$.

This means that if t and b are constant or depend only on k, then the (t,k)-sequence becomes a low-discrepancy sequence. Note that a smaller value of t gives a lower discrepancy asymptotically. Thus, a $(0,k)$-sequence can be said as the best in this sense. We should notice that any subset of $s < k$ coordinates of $(0,k)$-sequences constitutes $(0,s)$-sequences.

3.2 Generalized Faure Sequences

Niederreiter presented a general construction principle for (t,k)-sequences as follows: Let $k \geq 1$ and $b \geq 2$ and $B = \{0,1,...,b-1\}$. Accordingly, we define

(i) a commutative ring R with identity and $\mathrm{card}(R) = b$;

(ii) bijections $\psi_j : B \to R$ for $j = 1,2,...,$ with $\psi_j(0) = 0$ for all sufficiently large j;

(iii) bijections $\lambda_{hi} : R \to B$ for $h = 1,2,...,k$ and $i = 1,2,...,$ with $\lambda_{hi}(0) = 0$ for $1 \leq h \leq k$ and all sufficiently large i;

(iv) elements $c_{ij}^{(h)} \in R$ for $1 \leq h \leq k, 1 \leq i, 1 \leq j$, where for fixed h and j we have $c_{ij}^{(h)} = 0$ for all sufficiently large i.

For $n = 0,1,2,...,$ write $n = \sum_{r=1}^{\infty} a_r(n) b^{r-1}$ with $a_r(n) \in B$. For $h = 1,...,k$, set the h-th coordinate of the point X_n in $[0,1]^k$ as

$$X_n^{(h)} = \sum_{i=1}^{\infty} x_{ni}^{(h)} b^{-i},$$

where

$$x_{ni}^{(h)} = \lambda_{hi}\left(\sum_{j=1}^{\infty} c_{ij}^{(h)} \psi_j(a_j(n))\right) \in B$$

for $1 \leq h \leq k, 1 \leq i,$ and $0 \leq n$. We call $C^{(h)} = (c_{ij}^{(h)})$ the *generator matrix* for the h-th coordinate of a (t,k)-sequence.

We now describe how to construct such generator matrices so that we obtain low-discrepancy sequences called generalized Niederreiter sequences (see Tezuka [19]) [3]. The construction is based on $GF\{b,z\}$, i.e., the formal Laurent series expansions over the finite field $GF(b)$, where b is a prime power. Denote $S(z) \in GF\{b,z\}$ by

$$S(z) = \sum_{j=w}^{\infty} a_j z^{-j},$$

where all $a_j \in GF(b)$ and w is an arbitrary integer. Hereafter, we use the following notations: $[S(z)]$ denotes the polynomial part of $S(z)$ and $[S(z)]_{p(z)} \stackrel{\mathrm{def}}{=} [S(z)] \pmod{p(z)}$ with $0 \leq \deg([S(z)]_{p(z)}) < \deg(p(z))$.

[3] In 1995, Niederreiter and Xing [9] presented a new construction method of low-discrepancy sequences based on algebraic function fields. Today, their sequences are known as the "theoretically" best, but there are several implementation issues to be overcome before these sequences become available for practical use.

Let k polynomials $p_1(z), ..., p_k(z)$ be pairwise coprime and let $e_h = \deg(p_h) \geq 1$ for $1 \leq h \leq k$. For $m \geq 1, 1 \leq h \leq k$, and $j \geq 1$, consider the expansion

$$\frac{y_{hm}(z)}{p_h(z)^j} = \sum_{r=w}^{\infty} a^{(h)}(j, m, r) z^{-r},$$

by which the elements $a^{(h)}(j, m, r) \in GF(b)$ are determined. Here $w \leq 0$ may depend on h, j, m, and each $y_{hm}(z)$ is a polynomial such that the residue polynomials $[y_{hm}(z)]_{p_h(z)}, (j-1)e_h \leq m-1 < je_h$, are linearly independent over $GF(b)$ for any $j > 0$ and $1 \leq h \leq k$. Then define

$$c_{mr}^{(h)} = a^{(h)}(m_h + 1, m, r)$$

for $1 \leq h \leq k, m \geq 1$, and $r \geq 1$, where $m_h = [(m-1)/e_h]$.

Tezuka [19] proved the following theorem:

Theorem 6. *If an integer $t \geq 0$ satisfies $t \geq \sum_{h=1}^{k} \deg(p_h) - k$, then the generalized Niederreiter sequence becomes a (t, k)-sequence in base b.*

Remark 1. Faure sequences are $(0, k)$-sequences in a prime base $b \geq k$ obtained from generalized Niederreiter sequences such that all $y_{hm}(z) = 1$.

This remark motivates the following definition [18,19]:

Definition 7. *Generalized Faure sequences are defined as $(0, k)$-sequences obtained from generalized Niederreiter sequences.*

In practice, we choose the base b to be prime, for which we have the following matrix representation for all generator matrices $C^{(h)}, 1 \leq h \leq k$:

$$C^{(h)} = A^{(h)} P^{h-1}, \tag{4}$$

where $A^{(h)}, 1 \leq h \leq k$, are nonsingular lower triangular matrices over $GF(b)$ and P is the Pascal matrix whose (i, j) element is equal to $\binom{j-1}{i-1}$. The original Faure sequences correspond to the case in which $A^{(h)} = I$ for all h.

4 Applications to Finance

In this section, we apply quasi-Monte Carlo methods to a problem related to pricing Mortgage Backed Securities (MBS) originally described by Paskov [15]. In the experiment, we used the original Faure sequence and a generalized Faure sequence with base $b = 367$. More precisely, nonsingular lower triangular matrices $A^{(h)}, 1 \leq h \leq k$, in equation (4) for the generator matrices were chosen at random, and we omitted the first 100000 points; that is to say, we used the points $X_i, i = 100001, 100002, ...$, of the sequence.

Mortgage-Backed Securities

MBS, the most popular fixed income derivatives, are a kind of interest-rate option, whose underlying asset is a pool of residential mortgage portfolios [4]. They have a critical feature of prepayment privileges, because householders can prepay their mortgages at any time. The integration problem associated with MBS is summarized as follows: We use the following notation:

r_k: the appropriate interest rate in month k

w_k: the percentage prepaid in month k

a_k: the remaining annuity after $361-k$ months

C: the monthly payment on the underlying mortgage pool

for $k = 1, 2, ..., 360$, where $a_k = 1+d_1+\cdots+d_1^{k-1}$ is constant with $d_1 = 1/(1+r_0)$ and r_0 is the current monthly interest rate. C is also constant. The variable r_k follows the discrete-time version of the following geometric Brownian motion:

$$\log r_k - \log r_{k-1} = (a - \frac{\sigma^2}{2})\Delta + \sigma dB,$$

where $\Delta = 1$ and dB is the normal random variable with mean zero and variance Δ. Here, we assume zero drift (i.e., $a = 0$) in order to make $E(r_k) = r_0$ for $k = 1, ..., 360$. Thus,

$$r_k = K_0 \exp(\sigma z_k) r_{k-1}, \text{ for } k = 1, 2, ..., 360,$$

where $z_k, k = 1, 2, ..., 360$, are independent standard normally distributed random variables, and $K_0 = \exp(-\sigma^2/2)$.

The prepayment model for the variables $w_k, k = 1, 2, ..., 360$, depends on the interest rate $r_k, k = 1, 2, ..., 360$, as follows:

$$w_k = K_1 + K_2 \arctan(K_3 r_k + K_4),$$

where K_1, K_2, K_3, and K_4 are given constants. In practice, the lower the interest rate is, the higher the prepayment rate becomes. Thus, the cash flow in month k is

$$M_k(z_1, ..., z_k) = C(1-w_1)\cdots(1-w_{k-1})(1-w_k+w_k a_{361-k}).$$

This is multiplied by the discount factor

$$d_k(z_1, ..., z_{k-1}) = \prod_{i=0}^{k-1} \frac{1}{1+r_i}.$$

We have the following total present value of MBS:

$$PV(z_1, ..., z_{360}) = \sum_{k=1}^{360} d_k(z_1, ..., z_{k-1}) M_k(z_1, ..., z_k).$$

What we want to compute is the expected value of the present value PV over all independent random variables $z_k, k = 1, ..., 360$. By using the inversion of

the normal distribution, we can formulate this problem as one of computing a multivariate integration over $[0,1]^{360}$:

$$\mathrm{E}(PV) = \int_{[0,1]^{360}} PV(u_1, ..., u_{360}) du_1 \cdots du_{360},$$

where $u_k = \mathrm{N}(z_k)$ for $k = 1, ..., 360$.

In this experiment, we used the parameter set $(r_0, K_1, K_2, K_3, K_4, \sigma) = (.00625, .24, .134, -261.17, 12.72, .2)$ from [20], where the expected value of PV is numerically computed as $143.0182 \times C$ by using more than one million sample paths. Figure 1 shows the convergence of Monte Carlo and quasi-Monte Carlo. The solid line (MBS.MC) shows the result obtained by the Monte Carlo method, while the two different dotted lines (MBS.faure and MBS.gfaure) show the results obtained by the two quasi-Monte Carlo methods. Here, we stress two important points: (1) Monte Carlo vs. quasi-Monte Carlo (generalized Faure), and (2) original Faure vs. generalized Faure. For comparison, if we look at 1000 samples, generalized Faure sequences converge to the correct value within an accuracy of 10^{-5}, that is, 143.0182 ± 0.00143. On the other hand, the standard deviation of PV computed from the first 1000 sample values of the Monte Carlo simulation is about 0.276. Thus, the 99% confidence interval is about $143.0182 \pm 2.575 * 0.276/\sqrt{1000} = 143.0182 \pm 0.0225$. Therefore, we can say that about 250 times speed-up was gained by quasi-Monte Carlo with the generalized Faure sequence for this problem. On the other hand, from the comparison between Monte Carlo and quasi-Monte Carlo with the original Faure, we see the performance of the convergence looks similar.

5 Conclusion

Concluding the paper, I would like to present the following two interesting questions:

MC vs. QMC

Why does QMC outperform MC so significantly for high-dimensional numerical integration associated with finance problems? One idea to explain this result is the "effective dimension" [15,17,21], which comes from the observation that in many finance problems a small number of variables (say, near-future interest rates) are very crucial for the payoff function, while all the others are much less important. Thus, it is thought that even if the nominal dimension is large, say 360 for finance problems, the effective dimension should be small. This sounds why QMC was so successful for finance problems with large dimensions. Several proposals for the formal and quantitative definitions of the effective dimension have been made by several authors, but, at this moment, no decisive one is yet available to clearly explain the QMC's excellent performance in finance.

Papageorgiou [11,13] recently found that QMC also works very well compared with MC for isotropic problems in Physics. And Tezuka[20] pointed out

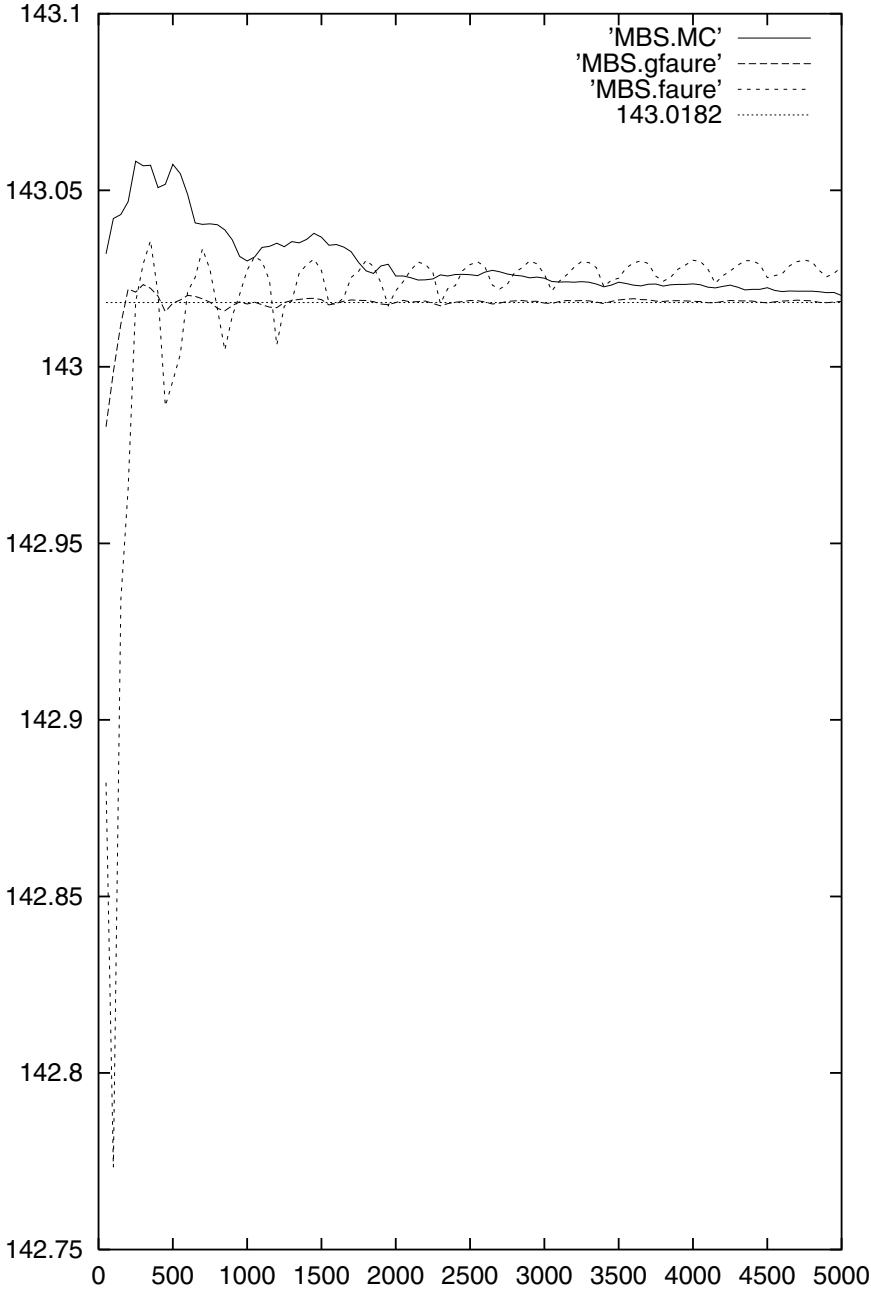


Fig. 1. Convergence of Monte Carlo and quasi-Monte Carlo methods for MBS (price vs. sample paths)

that there also exists an isotropic problem in finance for which QMC outperforms MC. In these cases, the idea of effective dimension does not work because the importance of all the variables is the same for the isotropic integrand. The problem of how to explain these results is another important research topic.

Faure vs. GFaure

Among many classes of low-discrepancy sequences, the practical behaviors of convergence rate are different for one from another, even if the theoretical convergence rates, i.e., the upper bounds of the discrepancy, are exactly the same. A striking example of this phenomenon is the difference between practical convergence behaviors of the original Faure and generalized Faure sequences as shown in the preceding section.

For generalized Faure sequences, the effect of the use of lower triangular matrices $A^{(h)}$ on the convergence rate has been theoretically and empirically studied by several researchers [5,6,10,12]. Assuming that the integrand is a fixed sufficiently smooth real function, their results imply that the expected integration error over all generalized Faure sequences becomes asymptotically

$$O\left(\frac{(\log N)^{(k-1)/2}}{N^{3/2}}\right).$$

We should notice that the denominator $N^{3/2}$ is much larger than N, which is well-known for ordinary QMC. Unfortunately, this is an asymptotic result. We need more elaborate on this for practical size of N and k. From another viewpoint, this can be interpreted as an existence theorem of a very good generalized Faure sequence for numerical multidimensional integration. Derandomization for finding such good sequences is very interesting in both theory and practice of low-discrepancy sequences.

Acknowledgments

I thank Professor Takayasu Ito for inviting me to the IFIP-TCS Conference in Sendai on August 17-19, 2000.

References

1. J. Beck and W. L. Chen, *Irregularities of Distribution*, Cambridge University Press, Cambridge, 1987.
2. M. Drmota and R. F. Tichy, *Sequences, Discrepancies and Applications*, Lecture Notes in Mathematics 1651, Springer-Verlag, Berlin, 1997.
3. F. J. Hickernell, Lattice Rules: How Well Do They Measure Up, in *Random and Quasi-Random Point Sets*, editted by P. Hellekalek and G. Larcher, Lecture Notes in Statistics, **138**, Springer, New York (1998), 109-166.
4. J. C. Hull, *Options, Futures, and Other Derivatives*, 3rd Edition, Prentice-Hall, New Jersey, 1997.

5. J. Matoušek, On the L_2-Discrepancy for Anchored Boxes, *Journal of Complexity*, **14** (1998), 527-556.
6. J. Matoušek, *Geometric Discrepancy: An Illustrated Guide,* Springer,1999.
7. H. Niederreiter, Quasi-Monte Carlo Methods and Pseudorandom Numbers, *Bull. Amer. Math. Soc.*, 84 (1978), 957-1041.
8. H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, CBMS-NSF Regional Conference Series in Applied Mathematics, No. 63, SIAM, 1992.
9. H. Niederreiter and C. Xing, Low-Discrepancy Sequences and Global Function Fields with Many Rational Places, *Finite Fields and Their Applications*, **2** (1996), 241-273.
10. A. Owen, Monte-Carlo Variance of Scrambled Net Quadrature, *SIAM J. Numer. Analysis*, **34** (1997), 1884-1910.
11. A. Papageorgiou, Fast Convergence of Quasi-Monte Carlo for a Class of Isotropic Integrals, to appear in *Math. Comp.*.
12. A. Papageorgiou and J. F. Traub, Beating Monte Carlo, *RISK*, **9** (June 1996), 63-65.
13. A. Papageorgiou and J. F. Traub, Faster Evaluation of Multidimensional Integrals, *Computers in Physics*, **11** (Nov/Dec 1997), 574-578.
14. S. H. Paskov, Average Case Complexity of Multivariate Integration for Smooth Functions, *Journal of Complexity*, **9** (1993), 291-312.
15. S. H. Paskov, New Methodologies for Valuing Derivatives, in *Mathematics of Derivative Securities*, edited by M. A. H. Dempster and S. Pliska, Isaac Newton Institute, Cambridge University Press, Cambridge UK (1997), 545-582.
16. S. H. Paskov and J. F. Traub, Faster Valuation of Financial Derivatives, *Journal of Portfolio Management*, **22**:1 (Fall 1995), 113-120.
17. I. H. Sloan and H. Woźniakowski, When are Quasi-Monte Carlo Algorithms Efficient for High Dimensional Integrals, *Journal of Complexity* **14** (1998), 1-33.
18. S. Tezuka, A Generalization of Faure Sequences and its Efficient Implementation, *Research Report RT-0105*,IBM Tokyo Research Laboratory, 1994.
19. S. Tezuka, *Uniform Random Numbers: Theory and Practice* , Kluwer Academic Publishers, Boston, 1995.
20. S. Tezuka, Financial Applications of Monte Carlo and Quasi-Monte Carlo Methods, in *Random and Quasi-Random Point Sets*, editted by P. Hellekalek and G. Larcher, Lecture Notes in Statistics, **138**, Springer, New York (1998), 303-332.
21. J. F. Traub and A. G. Werschulz, *Complexity and Information* , Cambridge Univ. Press, 1998.
22. G. Wasilkowski and H. Woźniakowski, The Exponent of Discrepancy is at Most 1.4778..., *Math. Comp.*, **66** (1997), 1125-1132.
23. H. Woźniakowski, Average Case Complexity of Multivariate Integration, *Bull. Amer. Math. Soc.* , **24** (1991), 185-194.

Fully Consistent Extensions of Partially Defined Boolean Functions with Missing Bits*

Endre Boros[1], Toshihide Ibaraki[2], and Kazuhisa Makino[3]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA. boros@rutcor.rutgers.edu

[2] Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan. ibaraki@i.kyoto-u.ac.jp

[3] Department of Systems and Human Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan. makino@sys.es.osaka-u.ac.jp

Abstract. In this paper we consider four different definitions for an extension of a partially defined Boolean function in which the input contains some missing bits. We show that, for many general and reasonable families of function classes, three of these extensions are mathematically equivalent. However we also demonstrate that such an equivalence does not hold for all classes.

1 Introduction

A *Boolean function*, or a *function* in short, is a mapping $f : \mathbb{B}^n \mapsto \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. Given a function f, a *Boolean vector* $x \in \mathbb{B}^n$ is called its *true vector*, if $f(x) = 1$, and its *false vector*, if $f(x) = 0$. Let us denote the set of true vectors of f by $T(f)$, and let $F(f) = \mathbb{B}^n \setminus T(f)$ denote the set of its false vectors. Let us denote by $\mathcal{C}_{all}$ the family of all Boolean functions $f : \mathbb{B}^n \mapsto \mathbb{B}$, and let us call any subfamily of $\mathcal{C}_{all}$ a *class*. We shall consider various classes of Boolean functions in the sequel, defined in many different ways.

A *partially defined Boolean function* (a *pdBf* in short) is defined by a pair of sets (T, F) such that $T, F \subseteq \mathbb{B}^n$. A Boolean function f is called an *extension* of the pdBf (T, F) if $T \subseteq T(f)$ and $F \subseteq F(f)$ hold, that is, if such an f *correctly classifies* all the vectors $a \in T$ and $b \in F$. Let us denote by $\mathcal{E}(T, F)$ the family of extensions of the pdBf (T, F). Evidently, the disjointness of the sets T and F is a necessary and sufficient condition for the existence of an extension $\mathcal{E}(T, F) \neq \emptyset$. It may not be evident, however, to find out if a given pdBf has an extension belonging to a particular class $\mathcal{C}$ of Boolean functions, or not. This problem has been studied in various fields such as learning theory, knowledge discovery, data mining and logical analysis of data [1,4,5,6,8,9,11,13].

* This research was partially supported by ONR (Grant N00014-92-J-1375), DARPA (Contract Number N66001-97-C-8537), and the Scientific Grants in Aid by the Ministry of Education, Science, Sports and Culture of Japan. The visit of the first author to Kyoto University was made possible by the grant (09044160) of the Ministry of Education, Science, Sports and Culture of Japan.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 257–272, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In practical cases, the fact that $\mathcal{C} \cap \mathcal{E}(T,F) = \emptyset$ might be due to some classification errors in the input. To correct this type of errors, provided that they are not in a large number, one can consider the optimization problem of finding the largest subsets $T^* \subseteq T$ and $F^* \subseteq F$ for which $\mathcal{E}(T^*,F^*) \cap \mathcal{C} \neq \emptyset$ holds. These problems have extensively been studied (e.g., in [8,11]) for a large variety of classes.

In this paper we shall consider another type of errors in the input, the case in which some data vectors are "incomplete" in the sense that some of their components are not available at the time of reading the input. Such missing information may either be due to some measurement errors at some earlier stages of data generation, or they are the results of data entry errors, or such lack of information might be due to the high cost of obtaining those.

To model such situations, let us consider the set $\mathbb{M} = \{0,1,*\}$, and let us interpret the asterisk components $*$ of a vector $v \in \mathbb{M}^n$ as missing bits. Then, a *pdBf with missing bits* (or in short a *pBmb*) can be defined as a pair $(\tilde{T},\tilde{F})$, where $\tilde{T},\tilde{F} \subseteq \mathbb{M}^n$. Given a pBmb, it is possible to consider more than one notion of extensions f, depending on how to interpret $*$'s in the extensions; in this paper, we give four different definitions, two of which have already been discussed in [9]. We then prove for many important classes of functions that three of these definitions are equivalent. However, it is also demonstrated that this equivalence does not hold for all classes.

2 Extensions of pBmbs

For a vector $v \in \mathbb{M}^n$, let us introduce the notations $ON(v) = \{j \,|\, v_j = 1,\ j = 1,2,\ldots,n\}$ and $OFF(v) = \{j \,|\, v_j = 0,\ j = 1,2,\ldots,n\}$. For a subset $\tilde{A} \subseteq \mathbb{M}^n$, let $S(\tilde{A}) = \{(v,j) | v \in \tilde{A},\ v_j = *\}$ be the collection of all missing bits of the vectors in $\tilde{A}$. If $\tilde{A}$ is a singleton $\{v\}$, we shall also write $S(v)$ instead of $S(\{v\})$. Clearly, $\mathbb{B}^n \subseteq \mathbb{M}^n$, and $v \in \mathbb{B}^n$ holds if and only if $S(v) = \emptyset$. Let us consider a binary assignment $\alpha \in \mathbb{B}^Q$ to a subset $Q \subseteq S(\tilde{A})$ of the missing bits. Then v^α denotes the vector obtained from $v \in \tilde{A}$ by replacing the $*$ components which belong to Q by the binary values assigned by α:

$$v_j^\alpha = \begin{cases} v_j & \text{if } (v,j) \notin Q \\ \alpha(v,j) & \text{if } (v,j) \in Q. \end{cases}$$

Let $\tilde{A}^\alpha$ denote the set $\{v^\alpha \mid v \in \tilde{A}\}$. For example, for the set $\tilde{A} = \{u = (1,*,0,1),\ v = (0,1,*,*),\ w = (1,1,*,0)\} \subset \mathbb{M}^4$ we have $S(\tilde{A}) = \{(u,2),\ (v,3),\ (v,4),\ (w,3)\}$. If $Q = \{(u,2),(v,4)\}$, an assignment $(\alpha(u,2),\alpha(v,4)) = (1,0) \in \mathbb{B}^Q$ yields $\tilde{A}^\alpha = \{u^\alpha = (1,1,0,1),\ v^\alpha = (0,1,*,0),\ w^\alpha = (1,1,*,0)\}$.

To a pBmb $(\tilde{T},\tilde{F})$ we shall always associate the set $S = S(\tilde{T} \cup \tilde{F})$ of its missing bits. For a pBmb $(\tilde{T},\tilde{F})$ and an assignment $\alpha \in \mathbb{B}^S$, let $(\tilde{T}^\alpha,\tilde{F}^\alpha)$ be the pdBf defined by $\tilde{T}^\alpha = \{a^\alpha \,|\, a \in \tilde{T}\}$ and $\tilde{F}^\alpha = \{b^\alpha \,|\, b \in \tilde{F}\}$.

Let us call a pBmb $(\tilde{T},\tilde{F})$ *consistent* with respect to a class $\mathcal{C}$ of Boolean functions, if there exists an assignment $\alpha \in \mathbb{B}^S$ for which the pdBf $(\tilde{T}^\alpha,\tilde{F}^\alpha)$

has an extension in $\mathcal{C}$. A Boolean function $f \in \mathcal{E}(\tilde{T}^\alpha, \tilde{F}^\alpha) \cap \mathcal{C}$ will be called a *consistent extension* of $(\tilde{T}, \tilde{F})$ in the class $\mathcal{C}$.

Problem CE($\mathcal{C}$)
Input: A pBmb $(\tilde{T}, \tilde{F})$, where $\tilde{T}, \tilde{F} \subseteq \mathbb{M}^n$.
Question: Does $(\tilde{T}, \tilde{F})$ have a consistent extension in class $\mathcal{C}$?

Let us note that, in case $(\tilde{T}, \tilde{F})$ has a consistent extension, the output of CE($\mathcal{C}$) might not be unique, and at an important extreme end, it may occur that for every possible interpretations of the missing bits the obtained pdBf has an extension belonging to $\mathcal{C}$. Let us call a pBmb $(\tilde{T}, \tilde{F})$ *fully consistent* with the class $\mathcal{C}$ if this occurs, i.e. if $(\tilde{T}^\alpha, \tilde{F}^\alpha)$ has an extension in $\mathcal{C}$ for every $\alpha \in \mathbb{B}^S$ (the corresponding extensions may differ for different α's.)

Problem FC($\mathcal{C}$)
Input: A pBmb $(\tilde{T}, \tilde{F})$, where $\tilde{T}, \tilde{F} \subseteq \mathbb{M}^n$.
Question: Is $(\tilde{T}, \tilde{F})$ fully consistent with the class $\mathcal{C}$?

Let us remark that, unlike for problem CE($\mathcal{C}$), confirming a YES for problem FC($\mathcal{C}$) might become a computational burden because one may have to provide $2^{|S|}$ different extensions, for each possible assignment to the missing bits of $(\tilde{T}, \tilde{F})$, even if each extension $f \in \mathcal{E}(\tilde{T}^\alpha, \tilde{F}^\alpha) \cap \mathcal{C}$ has a small representation. For this reason, we shall consider a special case in which in fact all these extensions coincide. Let us call a Boolean function f a *robust extension* of a given pBmb $(\tilde{T}, \tilde{F})$ if

$$f(a^\alpha) = 1 \text{ and } f(b^\alpha) = 0 \text{ for all } a \in \tilde{T},\ b \in \tilde{F} \text{ and for all } \alpha \in \mathbb{B}^S.$$

The corresponding decision problem can be stated as follows.

Problem RE($\mathcal{C}$)
Input: A pBmb $(\tilde{T}, \tilde{F})$, where $\tilde{T}, \tilde{F} \subseteq \mathbb{M}^n$.
Question: Does $(\tilde{T}, \tilde{F})$ have a robust extension in class $\mathcal{C}$?

Let us denote by $\mathcal{E}(\tilde{T}, \tilde{F})$ the family of all robust extensions of a given pBmb $(\tilde{T}, \tilde{F})$.

Let us remark now that even in this special case, the computational verification of a YES may not be an easy problem. Consider, for instance, the case when the output function f is represented by a DNF. Then, verifying that $f(a^\alpha) = 1$ holds for a vector $a \in \tilde{T}$ and for all $\alpha \in \mathbb{B}^{S(a)}$ might be as difficult as the tautology problem, which is known to be co-NP-complete even if its input is restricted to 3-DNF-s (see [12]).[1]

[1] The tautology problem is to decide if a given DNF φ satisfies $\varphi \equiv 1$. This is the complement of satisfiability problem, which is to decide, given a CNF φ, if there exists a vector v for which $\varphi(v) = 1$.

We shall therefore consider a further special case, when such difficulties will not arise. Consider an elementary conjunction (i.e., term)

$$t(\mathbf{x}) = \bigwedge_{j \in P} x_j \bigwedge_{j \in N} \overline{x}_j$$

for some subsets $P, N \subseteq \{1, 2, \ldots, n\}$ with $P \cap N = \emptyset$. We shall call t a *robust term* with respect to a pBmb $(\tilde{T}, \tilde{F})$ and a vector $a \in \tilde{T}$, if $t(a^\alpha) = 1$ for all $\alpha \in \mathbb{B}^{S(a)}$ and $t(b^\beta) = 0$ for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(b)}$. Let us note that a term t is robust with respect to $(\tilde{T}, \tilde{F})$ and $a \in \tilde{T}$ if and only if $S(a) \cap (P \cup N) = \emptyset$, and $(ON(b) \cap N) \cup (OFF(b) \cap P) \neq \emptyset$ for all $b \in \tilde{F}$, conditions which are all easy to check. Let us then call a Boolean function f a *very robust extension* of $(\tilde{T}, \tilde{F})$, if it is a robust extension which can be represented by a disjunction of robust terms.

PROBLEM VR($\mathcal{C}$)	
Input:	A pBmb $(\tilde{T}, \tilde{F})$, where $\tilde{T}, \tilde{F} \subseteq \mathbb{M}^n$.
Question:	Does $(\tilde{T}, \tilde{F})$ have a very robust extension in class $\mathcal{C}$?

provided.

Let us denote by $\mathcal{E}^*(\tilde{T}, \tilde{F})$ the family of all very robust extensions of the pBmb $(\tilde{T}, \tilde{F})$.

Problems CE($\mathcal{C}$) and RE($\mathcal{C}$) and some related optimization problems have been considered extensively for various classes in [7,9]. In this paper we concentrate on the relations between FC($\mathcal{C}$), RE($\mathcal{C}$) and VR($\mathcal{C}$). It is quite immediate to see from the above definitions that very robust extensions are robust as well, and that pBmbs which have robust extensions in a given class $\mathcal{C}$ are also fully consistent with that class.

Somewhat surprisingly, we can show that for many very general families of classes $\mathcal{C}$, these three problems FC($\mathcal{C}$), RE($\mathcal{C}$) and VR($\mathcal{C}$) are equivalent. However such an equivalence does not hold for all classes. In fact, for certain classes $\mathcal{C}$, problem RE($\mathcal{C}$) is polynomially solvable, while FC($\mathcal{C}$) is co-NP-complete.

3 Classes of Boolean Functions

We shall assume in the sequel that Boolean functions (functions, in short) are represented either by an explicit algebraic form, or by an oracle. In either case, it is possible to compute the values of such a function for given input vectors. In each case in the sequel, we shall make clear what is the representation of the considered family of functions.

Let us call an elementary conjunction of literals a *term*. The most common representation we shall consider for a function f will be either a *disjunctive normal form* (or DNF in short), which is a disjunction of terms.

For two functions f and g, we shall write $f \leq g$, if $f(x) = 1$ always implies $g(x) = 1$, and $f < g$ if $f \leq g$ and $f \neq g$. For a Boolean expression A,

let us denote by $\overline{A} = 1 - A$ its *negation*. The components of the (unknown) vector $x = (x_1, ..., x_n)$ will be called Boolean *variables*, while variables and their complements together are called *literals*.

A term t is called an *implicant* of a function f if $t \leq f$, and it is a *prime implicant* if t is a maximal implicant, i.e., $t \leq f$ and no term t' exists such that $t < t' \leq f$. It is well-known that every Boolean function can be represented by the DNF formed by the disjunction of all of its prime implicants. It is also well-known that in general, there are many other DNFs representing the same function.

We shall consider many different classes of Boolean functions, whose definitions will be either via some representation independent functional properties, or by properties of some of the DNF representations, or via some other representations.

A large family of classes of the first type are the *transitive* or *generalized monotone* classes. Let us consider a partial order $\preceq$ of the vectors $\mathbb{B}^n$, and let us say that a function f is $\preceq$-*monotone*, if $f(x) \leq f(y)$ holds whenever $x \preceq y$. For a given partial order $\preceq$ on $\mathbb{B}^n$, let us denote by $\mathcal{C}_{\preceq}$ the family of all $\preceq$-monotone functions. Then, we shall call a class $\mathcal{C}$ *transitive*, if there exists a partial order $\preceq$ on $\mathbb{B}^n$ for which $\mathcal{C} = \mathcal{C}_{\preceq}$.

Most notable examples for transitive classes are the family of positive (also called *monotone*) functions, $\mathcal{C}_+ = \mathcal{C}_{\geq}$, where $f \in \mathcal{C}_+$ if $f(x) \geq f(y)$ holds whenever $x \geq y$ holds (componentwise), and the family $\mathcal{C}_{reg}$ of *regular* Boolean functions, where $\mathcal{C}_{reg} = \mathcal{C}_{\succcurlyeq}$ for the relation $\succcurlyeq$, defined by $x \succcurlyeq y$ if and only if $\sum_{j=1}^{k} x_j \geq \sum_{j=1}^{k} y_j$ for all $1 \leq k \leq n$.

Another frequently used partial order on the Boolean cube is a "tilted" monotone order. To an arbitrary vector $b \in \mathbb{B}^n$, we can associate a partial order $\geq_b$ of the Boolean cube $\mathbb{B}^n$ by defining that $v \geq_b w$ holds if and only if $v \oplus b \geq w \oplus b$ holds, where $\oplus$ denotes the exclusive-or operation (the componentwise mod 2 addition, e.g. $(1100) \oplus (0110) = (1010)$). In other words, $\geq_b$ is like the regular monotone order $\geq$ in which b plays the role of the zero-vector $(0, 0, ..., 0)$, and $\overline{b}$ is the maximum vector. The family of $\geq_b$-monotone functions will be denoted by $\mathcal{C}_{\geq_b}$. Thus, in particular $\mathcal{C}_+ = \mathcal{C}_{\geq_0}$ holds. Let us finally remark that the family $\mathcal{C}_{all}$ of all Boolean functions itself is a transitive class, corresponding to the "empty" partial order on $\mathbb{B}^n$.

Some other non-transitive classes, defined via a representation independent property can be obtained by taking the union of various transitive classes. For instance, a function f is called *unate* if it is $\geq_b$-monotone for some vector $b \in \mathbb{B}^n$. The family of unate functions, hence is the union of all the $\geq_b$-monotone classes, $\mathcal{C}_{unate} = \bigcup_{b \in \mathbb{B}^n} \mathcal{C}_{\geq_b}$.

Other examples for classes defined via a representation independent property include the family of *self-dual* functions $\mathcal{C}_{SD}$, consisting of functions f for which $f = f^d$, where the *dual* f^d of a Boolean function f is defined by $f(x_1, ..., x_n) = \overline{f}(\overline{x}_1, ..., \overline{x}_n)$. Similarly, the family of *dual-minor* functions $\mathcal{C}_{D\text{-}minor}$ consists of the functions satisfying the inequality $f \leq f^d$, while the class of *dual-major* functions $\mathcal{C}_{D\text{-}major}$ is formed by the functions satisfying $f \geq f^d$.

Another large family of classes, the so called *DNF-classes* are defined via their DNF representation. Let us consider a family of terms $\mathbb{T}$, and let us define the corresponding DNF-class $\mathcal{C}_{\mathbb{T}}$ by $\mathcal{C}_{\mathbb{T}} = \{f(x) = \bigvee_{t\in\mathcal{S}} t(x) \,|\, \mathcal{S} \subseteq \mathbb{T}\}$ as the collection of all Boolean functions formed by the disjunction of a subset of terms from $\mathbb{T}$. For example, if $\mathbb{T}$ consists of all terms of degree at most k (elementary conjunctions involving at most k literals), then the corresponding DNF-class is the family of the so-called k-DNFs, $\mathcal{C}_{k\text{-}DNF}$. The special cases of linear functions ($\mathcal{C}_{1\text{-}DNF}$) and quadratic functions ($\mathcal{C}_{2\text{-}DNF}$) should, in particular be mentioned. Another notable example for a DNF-class is the family of *Horn* functions, $\mathcal{C}_{Horn}$. A Boolean function f is called *Horn*, if it can be represented by a DNF in which every term involves at most one negative variable. In other words, if $\mathbb{T}$ is the family of terms involving at most one negative variable, then Horn functions form the corresponding DNF-class, $\mathcal{C}_{\mathbb{T}} = \mathcal{C}_{Horn}$.

Let us remark that DNF-classes $\mathcal{C}_{\mathbb{T}}$ for which $\mathbb{T}$ is closed under consensus (for definition see Section 5) will play a special role due to the property that, for such a DNF, all its prime implicants must also belong to $\mathbb{T}$. Among consensus closed classes we should mention 2-DNFs, Horn functions, and $\geq_b$-monotone functions.

Given the Boolean functions f and g, we shall call g a *minor* of f, and will denote it by $g \sqsubseteq f$, if g can be represented by a disjunction of some of the prime implicants of f. Let us then call a class $\mathcal{C}$ *minor closed* if $f \in \mathcal{C}$ and $g \sqsubseteq f$ imply $g \in \mathcal{C}$. Minor closed classes include, in particular, all consensus closed DNF classes, and unions of those, such as renamable Horn functions, unate functions, q-Horn functions (see e.g., [3]), etc.

One other important class of functions, the class $\mathcal{C}_{TH}$ of *threshold* functions, is defined usually by a different representation. A Boolean function f is called *threshold* if there exist real numbers w_1, ..., w_n and w_0 such that $f(x) = 1$ if and only if the inequality $\sum_{j=1}^{n} w_j x_j \geq w_0$ holds. In other words, f is threshold exactly when the sets $T(f)$ and $F(f)$, viewed as point sets in the Euclidean space $\mathbb{R}^n$, are linearly separable. Of course, threshold functions could also be represented by DNFs (or CNFs), but for most threshold functions such a representation would be much less efficient computationally.

4 Equivalencies between RE($\mathcal{C}$) and FC($\mathcal{C}$)

In this section we shall show a series of results claiming, somewhat surprisingly, the equivalence of problems RE($\mathcal{C}$) and FC($\mathcal{C}$), under some widely applicable conditions. Let us remark here that two decision problems are equivalent if they have the same output (YES or NO) for all possible input. Equivalent decision problems are of course also equivalent computationally.

Let us also note that due to the space limitations we could not include all the proofs here, and we refer the reader to [10] for the missing details.

Let us first consider those classes $\mathcal{C}$ of Boolean functions which are closed under conjunction and disjunction; i.e., $f \wedge g \in \mathcal{C}$ and $f \vee g \in \mathcal{C}$ for all $f, g \in \mathcal{C}$.

Theorem 1. *Let us assume that the class $\mathcal{C}$ of Boolean functions is closed under conjunction and disjunction. Then problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent.*

Proof. Since the existence of a robust extension always implies full consistency, we only show the opposite implication. Assuming that the pBmb $(\tilde{T}, \tilde{F})$ is fully consistent with $\mathcal{C}$, we show that $(\tilde{T}, \tilde{F})$ has also a robust extension.

By the assumption, for every pair $\alpha \in \mathbb{B}^{S(\tilde{T})}$ and $\beta \in \mathbb{B}^{S(\tilde{F})}$, there exists an extension $f_{\alpha,\beta} \in \mathcal{C} \cap \mathcal{E}(\tilde{T}^\alpha, \tilde{F}^\beta)$. Let us then consider the Boolean function f defined by

$$f = \bigvee_{\alpha \in \mathbb{B}^{S(\tilde{T})}} \Big(\bigwedge_{\beta \in \mathbb{B}^{S(\tilde{F})}} f_{\alpha,\beta} \Big).$$

We claim that f is a robust extension of $(\tilde{T}, \tilde{F})$ in the class $\mathcal{C}$. First, $f \in \mathcal{C}$ follows from our assumption that $\mathcal{C}$ is closed under conjunction and disjunction.

To see that f is a robust extension of $(\tilde{T}, \tilde{F})$, let us first consider a vector $a \in \tilde{T}$ and an arbitrary assignment $\alpha^* \in \mathbb{B}^{S(\tilde{T})}$. Since $a^{\alpha^*} \in \tilde{T}^{\alpha^*}$, we have $\bigwedge_{\beta \in \mathbb{B}^{S(\tilde{F})}} f_{\alpha^*,\beta}(a^{\alpha^*}) = 1$ by the fact that $f_{\alpha^*,\beta} \in \mathcal{E}(\tilde{T}^{\alpha^*}, \tilde{F}^\beta)$ for all $\beta \in \mathbb{B}^{S(\tilde{F})}$. Thus $f(a^{\alpha^*}) = 1$ is implied, for all $\alpha^* \in \mathbb{B}^{S(\tilde{T})}$.

Analogously, for a vector $b \in \tilde{F}$ and an assignment $\beta^* \in \mathbb{B}^{S(\tilde{F})}$ we can observe first that $f_{\alpha,\beta^*}(b^{\beta^*}) = 0$ holds for all $\alpha \in \mathbb{B}^{S(\tilde{T})}$, implied again by $f_{\alpha,\beta^*} \in \mathcal{E}(\tilde{T}^\alpha, \tilde{F}^{\beta^*})$. Thus, in this case $\bigwedge_{\beta \in \mathbb{B}^{S(\tilde{F})}} f_{\alpha,\beta}(b^{\beta^*}) = 0$ follows for all $\alpha \in \mathbb{B}^{S(\tilde{T})}$, implying hence $f(b^{\beta^*}) = 0$, for all $\beta^* \in \mathbb{B}^{S(\tilde{F})}$.

These two observations then show that f is indeed a robust extension of $(\tilde{T}, \tilde{F})$. □

It is easy to see that transitive classes are closed under conjunction and disjunction (and even more, a class is transitive if and only if it is closed under conjunction and disjunction, see [2]) and hence the following corollary is implied by the above theorem:

Corollary 1. *Problems* $\mathrm{RE}(\mathcal{C})$ *and* $\mathrm{FC}(\mathcal{C})$ *are equivalent for all transitive classes, including* $\mathcal{C}_{all}$, $\mathcal{C}_+$, $\mathcal{C}_{regular}$ *and* $\mathcal{C}_{\geq_b}$ *for all* $b \in \mathbb{B}^n$.

Let us consider next certain lattice like transitive relations. We shall say that a partial order $\succeq$ on $\mathbb{B}^n$ is *cube-lattice like* if there is a unique $\succeq$-maximum and a unique $\succeq$-minimum in any subcube of $\mathbb{B}^n$, or equivalently, if for every term t, there are unique vectors $u, v \in T(t)$ such that $u \succeq w \succeq v$ holds for all $w \in T(t)$. Let us note that all partial orders mentioned in the previous section (e.g., $\geq_b$ for $b \in \mathbb{B}^n$, $\succcurlyeq$, etc.) are cube-lattice like, and there are many others. For instance, an arbitrary permutation of the 2^n vertices of $\mathbb{B}^n$, viewed as a linear order, is cube-lattice like.

For vectors $v, w \in \mathbb{M}^n$, we write $v \approx w$ if there is an assignment $\alpha \in \mathbb{B}^{S(\{v,w\})}$ such that $v^\alpha = w^\alpha$, and we say that v is *potentially identical* with w. For example, if $v = (1, 0, *, 1, *)$ and $w = (1, *, 0, 1, *)$ then $v \approx w$ holds.

To every vector $a \in \mathbb{M}^n$ we can associate the subcube $B(a) = \{u \in \mathbb{B}^n | u \approx a\} = \{a^\alpha | \alpha \in \mathbb{B}^{S(a)}\}$ of $\mathbb{B}^n$ consisting of all Boolean vectors one can obtain from a by assigning binary values to its missing bits. Given a vector $a \in \mathbb{M}^n$ and a cube-lattice like partial order $\succeq$ on $\mathbb{B}^n$, let us denote by $a^+ \in B(a)$ (resp.,

$a^- \in B(a)$) the unique $\succeq$-maximal vector (resp., the unique $\succeq$-minimal vector) in the subcube $B(a)$. Furthermore, for any subset $S \subseteq \mathbb{M}^n$ let $S^+ = \{a^+ | a \in S\}$ and $S^- = \{a^- | a \in S\}$ denote the corresponding subsets of Boolean vectors.

With these notations, we can state the following generalization of [9, Lemma 1]:

Lemma 1. *If $\mathcal{C}$ is an arbitrary subfamily of a transitive class $\mathcal{C}_{\succeq}$ with a cube-lattice like partial order $\succeq$, then a pBmb $(\tilde{T}, \tilde{F})$ has a robust extension in $\mathcal{C}$ if and only if the pdBf $(\tilde{T}^-, \tilde{F}^+)$ has an extension in $\mathcal{C}$.*

Proof. Since $\tilde{T}^- = \tilde{T}^{\alpha^*}$ for some $\alpha^* \in \mathbb{B}^{S(\tilde{T})}$, and $\tilde{F}^+ = \tilde{F}^{\beta^*}$ for some $\beta^* \in \mathbb{B}^{S(\tilde{F})}$, it follows that any robust extension of $(\tilde{T}, \tilde{F})$ will be an extension of $(\tilde{T}^-, \tilde{F}^+)$, by the definition of a robust extension.

To see the reverse direction, let us assume that $f \in \mathcal{E}(\tilde{T}^-, \tilde{F}^+) \cap \mathcal{C}$. Since all functions in $\mathcal{C}$ are $\succeq$-monotone, and since $a^\alpha \succeq a^-$ holds, we have $f(a^\alpha) \geq f(a^-) = 1$ implied for all $a \in \tilde{T}$ and $\alpha \in \mathbb{B}^{S(\tilde{T})}$. Similarly, $b^\beta \leq b^+$ and $f(b^+) = 0$ implies $f(b^\beta) \leq f(b^+) = 0$ for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(\tilde{F})}$. Hence, this function f is also a robust extension of $(\tilde{T}, \tilde{F})$ in $\mathcal{C}$. □

This lemma immediately implies the following statement.

Theorem 2. *If the class $\mathcal{C}$ is a subfamily of a transitive class $\mathcal{C}_{\succeq}$ with a cube-lattice like partial order $\succeq$, then problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent.*

Proof. Indeed, if the pBmb $(\tilde{T}, \tilde{F})$ is fully consistent with the class $\mathcal{C}$, then the pdBf $(\tilde{T}^-, \tilde{F}^+)$ has an extension $f \in \mathcal{C} \cap \mathcal{E}(\tilde{T}^-, \tilde{F}^+)$. This f will then be a robust extension of $(\tilde{T}, \tilde{F})$ in $\mathcal{C}$ by Lemma 1. The converse direction is obvious by the definitions. □

Corollary 2. *For any subfamily $\mathcal{C}$ of $\mathcal{C}_+$, $\mathcal{C}_{regular}$ and $\mathcal{C}_{\geq_b}$ for any $b \in \mathbb{B}^n$, problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent.*

Let us next consider DNF-classes.

Theorem 3. *Problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent for all DNF-classes $\mathcal{C} = \mathcal{C}_{\mathbb{T}}$.*

Proof. Let us only show that fully consistency implies the existence of a robust extension, since the converse direction is immediate from the definition.

Observe first that, given a true vector $a \in \tilde{T}$ and an assignment $\alpha \in \mathbb{B}^{S(a)}$, each false vector $b \in \tilde{F}$ has a unique assignment $\beta = \beta(\alpha) \in \mathbb{B}^{S(b)}$ minimizing the Hamming distance between the Boolean vectors a^α and b^β.

Let us fix an arbitrary vector $a \in \tilde{T}$ and an assignment $\alpha \in \mathbb{B}^{S(\tilde{T})}$, and define $\beta^* \in \mathbb{B}^{S(\tilde{F})}$ as the unique assignment which coincides with $\beta(\alpha) \in \mathbb{B}^{S(b)}$ for all $b \in \tilde{F}$. Such an assignment obviously can be constructed by concatenating the $\beta(\alpha)$ assignments for $b \in \tilde{F}$, since the sets $S(b)$ for $b \in \tilde{F}$ are pairwise disjoint.

Since $(\tilde{T}, \tilde{F})$ is fully consistent with $\mathcal{C}_{\mathbb{T}}$ by our assumption, there exists a Boolean function $g \in \mathcal{C}_{\mathbb{T}} \cap \mathcal{E}(\tilde{T}^\alpha, \tilde{F}^{\beta^*})$. Since a^α is a true vector of such an extension, g must have a term $t_{a,\alpha} \in \mathbb{T}$ for which $t_{a,\alpha}(a^\alpha) = 1$.

We claim that $t_{a,\alpha}(b^\beta) = 0$ holds for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(\tilde{F})}$. To see this, let us observe that for every vector $b \in \tilde{F}$ there must be a literal in $t_{a,\alpha}$ at which b^{β^*} and a^α are different, since otherwise $t_{a,\alpha}(b^{\beta^*}) = t_{a,\alpha}(a^\alpha) = 1$ would follow, contradicting the fact that g is an extension of the pdBf $(\tilde{T}^{\alpha^*}, \tilde{F}^{\beta^*})$. Then this literal does not correspond to any component of $S(b)$, otherwise we could switch its value in β^* to decrease the Hamming distance to a^α. Thus, this literal does not agree with any b^β for $\beta \in \mathbb{B}^{S(\tilde{F})}$, and hence the claim follows.

Therefore, the Boolean function defined by

$$f = \bigvee_{a \in \tilde{T}, \alpha \in \mathbb{B}^{S(\tilde{T})}} t_{a,\alpha}.$$

is a robust extension of $(\tilde{T}, \tilde{F})$ in $\mathcal{C}_{\mathbb{T}}$. Indeed, the equations $f(b^\beta) = 0$ hold for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(\tilde{F})}$ according to the above claim. Furthermore, for a true vector $a \in \tilde{T}$ and an arbitrary assignment $\alpha \in \mathbb{B}^{S(\tilde{T})}$ we have $f(a^\alpha) = 1$ implied by $t_{a,\alpha}(a^\alpha) = 1$. □

Corollary 3. *Problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent for* $\mathcal{C}_{all}$, $\mathcal{C}_{+}$, $\mathcal{C}_{k\text{-}DNF}$, $\mathcal{C}_{Horn}$.

Let us finally consider self-dual, dual-minor and dual-major functions.

Theorem 4. *Problems* RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are equivalent for* $\mathcal{C} = \mathcal{C}_{SD}$ *(resp.,* $\mathcal{C}_{D\text{-}minor}$ *and* $\mathcal{C}_{D\text{-}major}$*), if* $(*, *, \ldots, *) \notin \tilde{T} \cup \tilde{F}$ *(resp.,* $(*, *, \ldots, *) \notin \tilde{T}$ *and* $(*, *, \ldots, *) \notin \tilde{F}$*).*

Corollaries 1, 2, 3 and Theorem 4, together with the complexity results of RE($\mathcal{C}$) in [9], imply the following theorem.

Theorem 5. *Problem* FC($\mathcal{C}$) *is polynomially decidable for* $\mathcal{C} = \mathcal{C}_{all}, \mathcal{C}_{+}, \mathcal{C}_{regular}$. $\mathcal{C}^{+}_{k\text{-}DNF}$ *(for a constant* k*),* $\mathcal{C}_{k\text{-}DNF}$ *(for* $k = 1, 2$*),* $\mathcal{C}_{Horn}$, $\mathcal{C}^{(+)}_{SD}$, $\mathcal{C}^{(+)}_{D\text{-}minor}$ *and* $\mathcal{C}^{(+)}_{D\text{-}major}$*, where* $\mathcal{C}^{+}_{X}$ *denotes the class of positive functions in* $\mathcal{C}_X$*, while it is co-*NP*-complete for* $\mathcal{C} = \mathcal{C}_{k\text{-}DNF}$ *(for a constant* $k \geq 3$*).*

5 Very Robust Extensions

Very robust extensions play a computationally important role, since when they exists, they usually can efficiently be constructed. For instance, we shall show below that for most DNF-classes $\mathcal{C}$, if RE($\mathcal{C}$) can be solved in polynomial time, then a very robust extension can also be provided at the same time.

Let us recall that a term

$$t(x_1, \ldots, x_n) = \bigwedge_{j \in P} x_j \bigwedge_{j \in N} \overline{x}_j$$

is called a robust term with respect to $a \in \tilde{T}$ for a pBmb $(\tilde{T}, \tilde{F})$, if $t(a^\alpha) = 1$ for all $\alpha \in \mathbb{B}^{S(a)}$, and $t(b^\beta) = 0$ for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(b)}$. In other words, if and only if

$$P \subseteq ON(a) \text{ and } N \subseteq OFF(a), \text{ and} \tag{1}$$

$$P \cap OFF(b) \neq \emptyset \text{ or } N \cap ON(b) \neq \emptyset textforeveryvectorb \in \tilde{F}. \tag{2}$$

Since both of these conditions are independent of the assignments to missing bits of $(\tilde{T}, \tilde{F})$, checking these conditions is quite straightforward. Therefore, verifying that a given DNF is a very robust extension of a pBmb $(\tilde{T}, \tilde{F})$ can be done in linear time in the size of $(\tilde{T}, \tilde{F})$. It is also clear from the definition that in a very robust extension one never needs more than $|\tilde{T}|$ terms.

Furthermore, looking at conditions (1) and (2), it is easy to see that finding a robust term for a given pBmb $(\tilde{T}, \tilde{F})$ and vector $a \in \tilde{T}$ reduces to a feasibility question in an associated setcovering problem, and hence it is computationally tractable in most cases. The above immediately imply for instance the following statement.

Theorem 6. *Problem* VR($\mathcal{C}$) *can be solved in polynomial time for* $\mathcal{C} = \mathcal{C}_{all}$*, for* $\mathcal{C}_{\geq_b}$ *for with* $b \in \mathbb{B}^n$ *(thus in particular for* $\mathcal{C} = \mathcal{C}_+$*), for* $\mathcal{C} = \mathcal{C}_{Horn}$ *(and for all related classes, such as k-quasi Horn and k-quasi reverse Horn for any fixed k), and for* $\mathcal{C} = \mathcal{C}_{k\text{-}DNF}$ *with k fixed.*

Let us recall that a class $\mathcal{C}$ is minor closed, if $f \in \mathcal{C}$ and $g \sqsubseteq f$ imply $g \in \mathcal{C}$. To discuss properties of very robust extensions, we shall further recall the *consensus method* and some of its properties (see e.g., [14,15]). Given two terms $t = \bigwedge_{j \in P} x_j \bigwedge_{j \in N} \overline{x}_j$ and $t' = \bigwedge_{j \in P'} x_j \bigwedge_{j \in N'} \overline{x}_j$, we say that they are in *conflict* at variable x_j if $j \in (P \cap N') \cup (N \cap P')$ (i.e., if x_j appears in one and $\overline{x}_j$ appears in the other). If t and t' are in conflict at exactly one of the variables, then their *consensus* is a term $t'' = [t, t']$ defined by

$$t'' = \bigwedge_{j \in (P \setminus N') \cup (P' \setminus N)} x_j \bigwedge_{j \in (N \setminus P') \cup (N' \setminus P)} \overline{x}_j.$$

In other words, the consensus of t and t' is the conjunction of all the literals appearing in these terms, except the two, corresponding to the conflicting variable. It is easy to see that the inequality $t'' \leq t \vee t'$ holds, and that t'' is maximal for this property. This implies, in particular that if t and t' are implicants of the Boolean function f, then their consensus $t'' = [t, t']$ (when exists) is also an implicant of f. The *consensus method* is the algorithm, in which consensuses of implicants of a given DNF of f are formed as long as new implicants are generated. It is well-known (see e.g., [14]) that this method is complete in the sense that all prime implicants of f will be obtained in this way, starting from any DNF representation of f. Of course, all these notions and results can straightforwardly be translated for CNF representations using De Morgan's laws. The corresponding operation between clauses is known as *resolution*.

Returning to robust terms, we are now ready to prove the following statement.

Lemma 2. *If f is a robust extension of the pBmb $(\tilde{T}, \tilde{F})$, then for every vector $a \in \tilde{T}$, f has a prime implicant $t_a \leq f$, which is a robust term with respect to a.*

Proof. Let us consider an (arbitrary) DNF representation of f:

$$f = \bigvee_{i=1}^{m} t_i, \tag{3}$$

where

$$t_i = \bigwedge_{j \in P_i} x_j \bigwedge_{j \in N_i} \overline{x}_j. \tag{4}$$

Given a vector $a \in \tilde{T}$, we substitute $x_j = 1$ for variables with $j \in ON(a)$, and $x_j = 0$ for variables with $j \in OFF(a)$ into (3). Let $I \subseteq \{1, 2, ..., m\}$ denote the set of indices of those terms of (3) which do not vanish after this substitution. For terms t_i for $i \in I$, we have

$$P_i \cap OFF(a) = \emptyset \text{ and } N_i \cap ON(a) = \emptyset. \tag{5}$$

Let us denote the resulting DNF by $f' = \bigvee_{i \in I} t'_i$, where

$$t'_i = \bigwedge_{j \in P_i \setminus ON(a)} x_j \bigwedge_{j \in N_i \setminus OFF(a)} \overline{x}_j. \tag{6}$$

Since $1 = f(a^\alpha) = f'(a^\alpha)$ holds for all $\alpha \in \mathbb{B}^{S(a)}$, it follows that f' is the constant **1** function, and thus the only prime implicant **1** of f' can be obtained by a chain of consensuses, starting with the terms of f'. Let us note that if the terms t'_i and t'_k for some $i \neq k$, $i, k \in I$ have a consensus, then so do the terms t_i and t_k. Furthermore, the variables x_j, $j \in ON(a)$, appear only positively, and the variables x_j, $j \in OFF(a)$, appear only negatively in the resulting consensus $[t_i, t_k]$. Applying this observation recursively, we can repeat the same chain of consensuses which produced **1** from t'_i, $i \in I$, with the corresponding terms t_i, $i \in I$, yielding an implicant t'_a of f.

Clearly, t'_a involves only literals x_j for some $j \in ON(a)$ and $\overline{x}_j$ for some $j \in OFF(a)$, and thus t'_a satisfies condition (1). Therefore, by deleting some literals from t'_a if needed, we can obtain a prime implicant t_a of f still satisfying (1).

Let us note finally that any (prime) implicant of f must satisfy conditions (2) simply because f is a robust extension of $(\tilde{T}, \tilde{F})$. □

Theorem 7. *If $\mathcal{C}$ is a minor closed class, then problems* $\mathrm{RE}(\mathcal{C})$ *and* $\mathrm{VR}(\mathcal{C})$ *are equivalent.*

Proof. Since a very robust extension is also a robust extension, let us prove only the non-trivial direction of the stated equivalence.

Let us assume that $(\tilde{T}, \tilde{F})$ is a pBmb which has a robust extension $f \in \mathcal{E}(\tilde{T}, \tilde{F}) \cap \mathcal{C}$. According to Lemma 2, for every $a \in \tilde{T}$, f has a prime implicant $t_a \leq f$ which is a robust term of $(\tilde{T}, \tilde{F})$. We claim that the Boolean function

$$g = \bigvee_{a \in \tilde{T}} t_a \tag{7}$$

is a very robust extension of $(\tilde{T}, \tilde{F})$ in the class $\mathcal{C}$.

Clearly, $g \leq f$ is a minor of f by its definition, hence $g \in \mathcal{C}$ is implied by the facts that $\mathcal{C}$ is minor closed and $f \in \mathcal{C}$. Since $f \in \mathcal{E}(\tilde{T}, \tilde{F})$, the inequality $g \leq f$ also implies that $g(b^\beta) \leq f(b^\beta) = 0$ for all $b \in \tilde{F}$ and $\beta \in \mathbb{B}^{S(b)}$. Also, since g contains a robust term t_a for every $a \in \tilde{T}$, it follows that $g(a^\alpha) = 1$ holds for all $a \in \tilde{T}$ and $\alpha \in \mathbb{B}^{S(a)}$. This implies that g is a robust extension of the pBmb $(\tilde{T}, \tilde{F})$. Finally, since g contains only robust terms, it is a very robust extension of $(\tilde{T}, \tilde{F})$. □

Corollary 4. *Problems* RE($\mathcal{C}$) *and* VR($\mathcal{C}$) *are equivalent for* $\mathcal{C} = \mathcal{C}_{all}$, $\mathcal{C}_{\geq_b}$ *with* $b \in \mathbb{B}^n$ *(thus in particular* $\mathcal{C}_+$*),* $\mathcal{C}_{2\text{-}DNF}$, $\mathcal{C}_{Horn}$, $\mathcal{C}_{r\text{-}Horn}$, $\mathcal{C}_{unate}$, $\mathcal{C}_{q\text{-}Horn}$ *and* $\mathcal{C}_{D\text{-}minor}$.

The next corollary follows from Corollaries 3 and 4.

Corollary 5. *For the classes* $\mathcal{C} = \mathcal{C}_{all}, \mathcal{C}_{Horn}, \mathcal{C}_+$, *and* $\mathcal{C}_{2\text{-}DNF}$, *problems* VR($\mathcal{C}$), RE($\mathcal{C}$) *and* FC($\mathcal{C}$) *are all equivalent.*

Besides Theorem 6, we have the following complexity results from Corollary 4 and the results of RE($\mathcal{C}$) in [9].

Theorem 8. *Problem* VR($\mathcal{C}$) *is polynomially solvable for* $\mathcal{C} = \mathcal{C}_{D\text{-}minor}$, *while it is* NP*-hard for* $\mathcal{C} = \mathcal{C}_{r\text{-}Horn}$ *and* $\mathcal{C}_{unate}$.

6 Cases of Non-equivalence between FC($\mathcal{C}$), RE($\mathcal{C}$), and VR($\mathcal{C}$)

In this section we shall show that problems FC($\mathcal{C}$), RE($\mathcal{C}$) and VR($\mathcal{C}$) are not always equivalent, despite the many quite general equivalences we have shown in the previous sections. We first give several classes $\mathcal{C}$ for which FC($\mathcal{C}$) and RE($\mathcal{C}$) are not equivalent, followed by a class for which RE($\mathcal{C}$) and VR($\mathcal{C}$) are not equivalent.

First, one might think that Theorem 1 could be generalized to prove the equivalence of RE($\mathcal{C}$) and FC($\mathcal{C}$) for classes closed under conjunction (but not necessarily closed under disjunction). This, however, is not the case, as the following simple example shows. Let us consider the class $\mathcal{C}^*$ consisting of functions f for which $f(v)f(w) = 0$ holds for all pairs of vectors $v, w \in \mathbb{B}^n$ which are at Hamming distance 1. Clearly, this class $\mathcal{C}^*$ is closed under conjunction. Let

us now consider the pBmb $(\tilde{T}, \tilde{F})$ given by $\tilde{T} = \{(1, *)\}$ and $\tilde{F} = \emptyset$. Since the equation $f(1,0) = f(1,1) = 1$ must hold for any robust extension f of $(\tilde{T}, \tilde{F})$, f does not belong to $\mathcal{C}^*$. Therefore, $(\tilde{T}, \tilde{F})$ has no robust extension in $\mathcal{C}^*$. However, $f = x_1 x_2 \in \mathcal{C}^*$ is an extension of the pdBf $(\{(1,1)\}, \emptyset)$ and $g = x_1 \bar{x}_2 \in \mathcal{C}^*$ is an extension of the pdBf $(\{(1,0)\}, \emptyset)$. These imply that $(\tilde{T}, \tilde{F})$ is fully consistent with $\mathcal{C}^*$.

Let us demonstrate next that, for the class of threshold functions $\mathcal{C}_{TH}$, problems RE($\mathcal{C}$) and FC($\mathcal{C}$) are not equivalent. Let us recall first that a pdBf (T, F) has a threshold extension, if and only if there exist $n+1$ real numbers $w_1, w_2, \ldots, w_n$ and w_0 such that:

$$\sum_{j=1}^{n} w_j a_j \ \geq \ w_0 \ \text{ for all } a \in T, \quad \textit{textand} \quad \sum_{j=1}^{n} w_j b_j \ < \ w_0 \ \text{ for all } b \in F. \tag{8}$$

It is well known that this condition is also equivalent to the disjointness of their respective convex hulls,

$$conv(T) \cap conv(F) = \emptyset, \tag{9}$$

where $conv(X)$ denotes the convex hull of the set X in the n-dimensional real space. It is also easy to see by the definitions that a pBmb $(\tilde{T}, \tilde{F})$ has a robust threshold extension if and only if

$$conv(\tilde{T}) \cap conv(\tilde{F}) = \emptyset, \tag{10}$$

where $conv(\tilde{X}) = conv(\cup_{\alpha \in \mathbb{B}^{S(\tilde{X})}} \tilde{X}^{\alpha})$ for a subset $\tilde{X} \subseteq \mathbb{M}^n$.

Let us now consider the pBmb $(\tilde{T}, \tilde{F})$ defined by

$$\tilde{T} = \left\{ \begin{pmatrix} *, 1, 1, 1 \end{pmatrix} \right\}, \quad \tilde{F} = \left\{ \begin{pmatrix} 1, 1, 1, 0 \end{pmatrix} \right\}.$$

The only one missing bit of $(\tilde{T}, \tilde{F})$ has two possible interpretations, yielding $T^1 = \{(1,1,1,1), (0,0,0,0)\}$ and $T^0 = \{0,1,1,1), (0,0,0,0)\}$. It is easy to verify that the threshold Boolean function defined by $5x_1 - 3x_2 - 3x_3 + 2x_4 \geq 0$ is an extension of the pdBf $(T^1, \tilde{F})$, and that $-5x_1 + 2x_2 + 2x_3 - 3x_4 \geq 0$ defines a threshold extension of $(T^0, \tilde{F})$. Hence, the pBmb $(\tilde{T}, \tilde{F})$ is fully consistent with $\mathcal{C}_{TH}$. However, $(\tilde{T}, \tilde{F})$ has no robust threshold extension by (10), since the fractional vector $(\frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3})$ belongs to the convex hulls of both $\tilde{T}$ and $\tilde{F}$.

We can also show the non-equivalence of FC($\mathcal{C}$) and RE($\mathcal{C}$) for the classes such as $\mathcal{C}_{unate}$ and $\mathcal{C}_{r\text{-}Horn}$.

In concluding this section, we demonstrate with the example $(\tilde{T}, \tilde{F})$ defined in the table below that RE($\mathcal{C}_{3\text{-}DNF}$) and VR($\mathcal{C}_{3\text{-}DNF}$) are not equivalent.

$$\tilde{T} = \left\{ \begin{pmatrix} 1, 1, *, 1, 1 \end{pmatrix} \right\}, \quad \tilde{F} = \left\{ \begin{pmatrix} 1, 1, 0, 1, 0 \end{pmatrix} \right\}.$$

This pBmb does not have a very robust 3-DNF extension, because the only robust term for $a = (1, 1, *, 1, 1)$ is the quartic term $t = x_1x_2x_4x_5$.

On the other hand, the 3-DNF

$$\phi = x_1x_2x_3 \vee \overline{x}_3x_4x_5$$

is a robust extension of $(\tilde{T}, \tilde{F})$, hence $\mathcal{E}(\tilde{T}, \tilde{F}) \cap \mathcal{C}_{3\text{-}DNF} \neq \emptyset$.

Let us remark that in fact, problem $\mathrm{VR}(\mathcal{C}_{3\text{-}DNF})$ is always polynomially decidable, while $\mathrm{RE}(\mathcal{C}_{3\text{-}DNF})$ is co-NP-complete (see e.g., [9]).

7 Complexity of $\mathrm{FC}(\mathcal{C}_{TH})$

We have already seen in Section 6 that problems $\mathrm{RE}(\mathcal{C}_{TH})$ and $\mathrm{FC}(\mathcal{C}_{TH})$ are not equivalent. It is known that $\mathrm{RE}(\mathcal{C}_{TH})$ is polynomially solvable (see e.g. [9]), and we can show below that problem $\mathrm{FC}(\mathcal{C}_{TH})$ is not only inequivalent, but has in fact a different complexity.

Theorem 9. *Problem* $\mathrm{FC}(\mathcal{C}_{TH})$ *is co-NP-complete, even if* $|S(a)| \leq 1$ *holds for all* $a \in \tilde{T} \cup \tilde{F}$.

Proof. First we show that $\mathrm{FC}(\mathcal{C}_{TH})$ belongs to co-NP. By (9), a pBmb $(\tilde{T}, \tilde{F})$ is not fully consistent with the class $\mathcal{C}_{TH}$ if and only if there exists an assignment $\alpha \in \mathbb{B}^S$ such that $conv(\tilde{T}^\alpha) \cap conv(\tilde{F}^\alpha) \neq \emptyset$. Therefore, $\mathrm{FC}(\mathcal{C}_{TH})$ is in co-NP, since the last condition can be checked in polynomial time (for instance by linear programming).

To prove the completeness, we reduce the following NP-complete problem to our problem (see e.g., [12]).

Problem Exact Cover	
Input:	A hypergraph $\mathcal{H} = (V, H)$ such that $V = \{1, 2, \ldots, n\}$ and $H = \{E_1, E_2, \ldots, E_m\}$, where $E \subseteq V$ for all $E \in H$.
Question:	Is there an $H^* \subseteq H$ which exactly covers V; i.e., for which $E \cap E' = \emptyset$ for all $E \neq E' \in H^*$ and $\bigcup_{E \in H^*} E = V$?

We may assume without loss of generality that any H^* which exactly covers V contains E_1. This does not affect the NP-hardness of the problem, as it can be seen easily, since we always can modify the input by including one more hyperedge, E_1, which is disjoint from all other hyperedges of H.

Let $V_1 = \{n+1, n+2, \ldots, n+m\}$ and $V_2 = \{n+m+1, n+m+2, \ldots, n+2m\}$ and let $W = V \cup V_1 \cup V_2$. We shall denote by $(R; S)$ the vector $v \in \mathbb{M}^W$ for which $ON(v) = R$ and $S(v) = \{(v, j) \mid j \in S\}$. (Then $OFF(v) = V \setminus (R \cup S)$; thus in particular, $v = (R; \emptyset)$ denotes a binary vector.) Let us define a pBmb $(\tilde{T}, \tilde{F})$ by the following $\tilde{T}, \tilde{F} \subseteq \mathbb{M}^W$.

$$\begin{aligned}
\tilde{T} &= \{a_{(1)} = (V \cup \{n+1\} \cup \{n+m+1\}; \emptyset)\} \\
&\quad \cup \{a_{(i)} = (\{n+m+i\}; \{n+i\}), i = 2, 3, \ldots, m\} \\
\tilde{F} &= \{b_{(0)} = (\emptyset; \emptyset)\} \cup \{b_{(1)} = (E_1 \cup \{n+1\} \cup V_2; \emptyset)\} \\
&\quad \cup \{b_{(i)} = (E_i \cup \{n+i\}; \emptyset), i = 2, 3, \ldots, m\}.
\end{aligned}$$

For this pBmb we have $|S(a)| \leq 1$ for all $a \in \tilde{T}$ and $S(\tilde{F}) = \emptyset$. Thus, we write simply $\tilde{F}$ instead of $\tilde{F}^{\alpha}$, in the sequel.

We claim that this $(\tilde{T}, \tilde{F})$ is not fully consistent with $\mathcal{C}_{TH}$ if and only if an exact cover $H^* \subseteq H$ exists. This will then imply the theorem.

First, we show the "only-if" part of the above claim. Let us assume that for an assignment $\alpha \in \mathbb{B}^{S(\tilde{T})}$ the pdBf $(\tilde{T}^{\alpha}, \tilde{F})$ has no threshold extension. It follows from (9) that there exist nonnegative real numbers θ_i $(i = 1, 2, \ldots, m)$ and η_i $(i = 0, 1, \ldots m)$ such that

$$\sum_{i=1}^{m} \theta_i = 1, \quad \sum_{i=0}^{m} \eta_i = 1, \quad \text{and} \quad \sum_{i=1}^{m} \theta_i a^{\alpha}_{(i)} = \sum_{i=0}^{m} \eta_i b_{(i)}. \tag{11}$$

By comparing the corresponding components on the two sides of the last equality of (11), we have

$$\eta_1 = \theta_i = \frac{1}{m} \text{ for } i = 1, 2, ..., m \tag{12}$$

$$\eta_i = \begin{cases} \frac{1}{m} \text{ if } \alpha(a_{(i)}, n+i) = 1, \\ 0 \text{ if } \alpha(a_{(i)}, n+i) = 0. \end{cases} \text{ for } i = 2, 3, \ldots, m \tag{13}$$

Moreover, $\eta_0 = 1 - \sum_{i=1}^{m} \eta_i \geq 0$ follows.

Let us define now a family $H^* \subseteq H$ by

$$H^* = \{E_i \,|\, \eta_i = \frac{1}{m},\ i = 1, 2, ..., m\},$$

Although the proof is omitted (see [10]), we can prove that H^* is an exact cover of H, which completes the "only-if" part of our claim.

For the "if" part, take an arbitrary exact cover $H^* \subseteq H$, and associate an assignment $\alpha \in \tilde{F}$ to it by defining

$$\alpha(a_{(i)}, n+i) = \begin{cases} 1 & \text{if } E_i \in H^* \\ 0 & \text{otherwise.} \end{cases}$$

It is then easy to see that with the nonnegative real numbers

$$\theta_i = \frac{1}{m}, \text{ for } i = 1, 2, ..., m,$$

$$\eta_i = \begin{cases} \frac{1}{m} \text{ if } E_i \in H^*, \\ 0 \quad \text{otherwise,} \end{cases} \text{ for } i = 1, 2, ..., m,$$

and $\eta_0 = 1 - \frac{|H^*|}{m}$, all equations in (11) hold. Hence $\tilde{T}^{\alpha}$ and $\tilde{F}$ are not linearly separable, which proves that $(\tilde{T}, \tilde{F})$ is not fully consistent with $\mathcal{C}_{TH}$. □

8 Conclusion

In this paper, we have considered the relations between problems FC($\mathcal{C}$), RE($\mathcal{C}$) and VR($\mathcal{C}$). We showed that for many general and reasonable families of classes such as $\mathcal{C} = \mathcal{C}_{all}$, $\mathcal{C}_{Horn}$, $\mathcal{C}_{+}$, and $\mathcal{C}_{2\text{-}DNF}$, these three problems are equivalent. We also demonstrated that such an equivalence does not hold for all classes $\mathcal{C}$. For instance, we showed that problem RE($\mathcal{C}_{TH}$) is polynomially solvable, while FC($\mathcal{C}_{TH}$) is co-NP-complete.

References

1. M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.
2. J.C. Bioch and T. Ibaraki. Lattice theoretic approaches to generalized monotone boolean functions and partially defined boolean functions. Manuscript, February 2000.
3. E. Boros, Y. Crama, and P.L. Hammer. Polynomial time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.
4. E. Boros, V. Gurvich, P.L. Hammer, T. Ibaraki, and A. Kogan. Decompositions of partially defined Boolean functions. *Discrete Applied Mathematics*, 62:51–75, 1995.
5. E. Boros, P.L. Hammer, T. Ibaraki, and A. Kogan. Logical analysis of numerical data. *Mathematical Programming*, 79:163–190, August 1997.
6. E. Boros, P.L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of logical analysis of data. *IEEE Transactions on Knowledge and Data Engineering*, to appear.
7. E. Boros, T. Ibaraki, and K. Makino. Monotone extensions of Boolean data sets. In M. Li and A. Maruoka, editors, *Algorithmic Learning Theory – ALT'97*, volume 1316 of *Lecture Notes in Artificial Intelligence*, pages 161–175, Berlin - New York, 1997. Springer Verlag.
8. E. Boros, T. Ibaraki, and K. Makino. Error-free and best-fit extensions of partially defined Boolean functions. *Information and Computation*, 140:254–283, 1998.
9. E. Boros, T. Ibaraki, and K. Makino. Logical analysis of binary data with missing bits. *Artificial Intelligence*, 107:219–263, 1999.
10. E. Boros, T. Ibaraki, and K. Makino. Fully consistent extensions of partially defined Boolean functions with missing bits. RUTCOR Research Report 9-99, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854-8003, USA, 1999 (http://rutcor.rutgers.edu/~rrr/1999.html).
11. Y. Crama, P.L. Hammer, and T. Ibaraki. Cause-effect relationships and partially defined Boolean functions. *Annals of Operations Research*, 16:299–326, 1988.
12. M.R. Garey and D.S. Johnson. *Computers and Intractability.* Freeman, New York, NY, 1979.
13. K. Makino, K. Hatanaka, and T. Ibaraki. Horn extensions of a partially defined Boolean function. *SIAM Journal on Computing*, 28:2168–2186, 1999.
14. W. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62:627–631, 1955.
15. S. Rudeanu. *Boolean Functions and Equations.* North-Holland, Amsterdam, 1974.

Characterization of Optimal Key Set Protocols
(Extended Abstract)

Takaaki Mizuki[1], Hiroki Shizuya[2], and Takao Nishizeki[1]

[1] Graduate School of Information Sciences, Tohoku University,
Aoba-yama 05, Aoba-ku, Sendai 980-8579, Japan
{mizuki,nishi}@ecei.tohoku.ac.jp
[2] Education Center for Information Processing, Tohoku University,
Kawauchi, Aoba-ku, Sendai 980-8576, Japan
shizuya@ecip.tohoku.ac.jp

Abstract. Using a random deal of cards to players and a computationally unlimited eavesdropper, all players wish to share a common one-bit secret key which is information-theoretically secure from the eavesdropper. This can be done by the so-called key set protocol. In this paper we give a necessary and sufficient condition for a key set protocol to be "optimal," that is, to succeed always in sharing a one-bit secret key.

1 Introduction

Suppose that there are k (≥ 2) players $P_1, P_2, \cdots, P_k$ and a passive eavesdropper, Eve, whose computational power is unlimited. All players wish to share a common one-bit secret key that is information-theoretically secure from Eve. Let C be a set of d distinct cards which are numbered from 1 to d. All cards in C are randomly dealt to players $P_1, P_2, \cdots, P_k$ and Eve. We call a set of cards dealt to a player or Eve a *hand*. Let $C_i \subseteq C$ be P_i's hand, and let $C_e \subseteq C$ be Eve's hand. We denote this *deal* by $\mathcal{C} = (C_1, C_2, \cdots, C_k; C_e)$. Clearly $\{C_1, C_2, \cdots, C_k, C_e\}$ is a partition of set C. We write $c_i = |C_i|$ for each $1 \leq i \leq k$ and $c_e = |C_e|$, where $|A|$ denotes the cardinality of a set A. Note that $c_1, c_2, \cdots, c_k$ and c_e are the sizes of hands held by $P_1, P_2, \cdots, P_k$ and Eve respectively, and that $d = \sum_{i=1}^{k} c_i + c_e$. We call $\gamma = (c_1, c_2, \cdots, c_k; c_e)$ the *signature* of deal $\mathcal{C}$. In this paper we assume that $c_1 \geq c_2 \geq \cdots \geq c_k$; if necessary, we rename the players. The set C and the signature γ are public to all the players and even to Eve, but the cards in the hand of a player or Eve are private to herself, as in the case of usual card games. This paper addresses protocols which make all the players share a common one-bit secret key information-theoretically securely using such a random deal of cards [2,3,4,5,6,10]. A reasonable situation in which such protocols are practically required is discussed in [4,6], and also the reason why we deal cards even to Eve is found there.

We consider a graph called a *key exchange graph*, in which each vertex i represents a player P_i and each edge (i, j) joining vertices i and j represents a pair of players P_i and P_j sharing a one-bit secret key $r_{ij} \in \{0, 1\}$. Refer to

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 273–285, 2000.
© Springer-Verlag Berlin Heidelberg 2000

[8] for the graph-theoretic terminology. A connected graph having no cycle is called a tree. If the key exchange graph is a tree, then all the players can share a common one-bit secret key $r \in \{0,1\}$ as follows: an arbitrary player chooses a one-bit secret key $r \in \{0,1\}$, and sends it to the rest of the players along the tree; when player P_i sends r to player P_j along an edge (i,j) of the tree, P_i computes the exclusive-or $r \oplus r_{ij}$ of r and r_{ij} and sends it to P_j, and P_j obtains r by computing $(r \oplus r_{ij}) \oplus r_{ij}$. For $k = 2$, Fischer, Paterson and Rackoff give a protocol to form a tree, i.e. a graph having exactly one edge, as the key exchange graph by using a random deal of cards [2]. Fischer and Wright extend this protocol for any $k \geq 2$, and formalize a class of protocols called "key set protocols," a formal definition of which will be given in the succeeding section [3, 6]. We say that a "key set protocol" *works for a signature* γ if the protocol always forms a tree as the key exchange graph for any deal $\mathcal{C}$ having the signature γ.

Let Γ_k be the set of all signatures of deals for k players, where the total number d of dealt cards is not fixed but takes any value. Furthermore, let Γ be the set of all signatures where the number k of players is taken over all values, that is,

$$\Gamma = \bigcup_{k=2}^{\infty} \Gamma_k.$$

Define sets W and L as follows:

$$W = \{\gamma \in \Gamma \mid \text{there is a key set protocol working for } \gamma\}; \text{ and}$$

$$L = \{\gamma \in \Gamma \mid \text{there is no key set protocol working for } \gamma\}.$$

Thus $\{W, L\}$ is a partition of set Γ. For $k = 2$, i.e. $\gamma \in \Gamma_2$, Fischer and Wright give a simple necessary and sufficient condition for $\gamma \in W$ [3]. For $k \geq 3$, the authors give a simple necessary and sufficient condition for $\gamma \in W$ [10]. (These necessary and sufficient conditions will be described in Section 2.5.)

One wishes to design a key set protocol which works for all signatures $\gamma \in W$, that is, always forms a tree as the key exchange graph for all deals $\mathcal{C}$ having any signature $\gamma \in W$. Such a protocol is said to be *optimal* for the class of key set protocols [3,6]. There exists an optimal key set protocol indeed: the "SFP protocol" given by Fischer and Wright is an example of an optimal key set protocol [3,6]. However, neither an optimal key set protocol other than the SFP protocol nor a characterization of optimal key set protocols has been known so far.

In this paper, using the condition for $\gamma \in W$ in [10], we give a complete characterization of optimal key set protocols, that is, we give a necessary and sufficient condition for a key set protocol to be optimal. Using the characterization, we can design many optimal key set protocols. Thus we show that not only the SFP protocol but also many others are optimal. Using these optimal protocols, one can produce trees of various shapes as a key exchange graph; some of them would be appropriate for efficient broadcast of a secret message. For example, one can produce a tree of a small radius, as we will show later in Section 4.

2 Preliminaries

In this section we explain the "key set protocol" formalized by Fischer and Wright, and present some of the known results on this protocol [2,3,6,10].

2.1 Key Set Protocol

We first define some terms. A *key set* $K = \{x, y\}$ consists of two cards x and y, one in C_i, the other in C_j with $i \neq j$, say $x \in C_i$ and $y \in C_j$. We say that a key set $K = \{x, y\}$ is *opaque* if $1 \leq i, j \leq k$ and Eve cannot determine whether $x \in C_i$ or $x \in C_j$ with probability greater than 1/2. Note that both players P_i and P_j know that $x \in C_i$ and $y \in C_j$. If K is an opaque key set, then P_i and P_j can share a one-bit secret key $r_{ij} \in \{0, 1\}$, using the following rule agreed on before starting a protocol: $r_{ij} = 0$ if $x > y$; $r_{ij} = 1$, otherwise. Since Eve cannot determine whether $r_{ij} = 0$ or $r_{ij} = 1$ with probability greater than 1/2, the secret key r_{ij} is information-theoretically secure. We say that a card x is *discarded* if all the players agree that x has been removed from someone's hand, that is, $x \notin (\bigcup_{i=1}^{k} C_i) \cup C_e$. We say that a player P_i *drops out* of the protocol if she no longer participates in the protocol. We denote by V the set of indices i of all the players P_i remaining in the protocol. Note that $V = \{1, 2, \cdots, k\}$ before starting a protocol.

The "key set protocol" has four steps as follows.

1. Choose a player P_s, $s \in V$, as a *proposer* by a certain procedure.
2. The proposer P_s determines in mind two cards x, y. The cards are randomly picked so that x is in her hand and y is not in her hand, i.e. $x \in C_s$ and $y \in (\bigcup_{i \in V - \{s\}} C_i) \cup C_e$. Then P_s proposes $K = \{x, y\}$ as a key set to all the players. (The key set is proposed just as a set. Actually it is sorted in some order, for example in ascending order, so Eve learns nothing about which card belongs to C_s unless Eve holds y.)
3. If there exists a player P_t holding y, then P_t accepts K. Since K is an opaque key set, P_s and P_t can share a one-bit secret key r_{st} that is information-theoretically secure from Eve. (In this case an edge (s, t) is added to the key exchange graph.) Both cards x and y are discarded. Let P_i be either P_s or P_t that holds the smaller hand; if P_s and P_t hold hands of the same size, let P_i be the proposer P_s. P_i discards all her cards and drops out of the protocol. Set $V := V - \{i\}$. Return to step 1.
4. If there exists no player holding y, that is, Eve holds y, then both cards x and y are discarded. Return to step 1. (In this case no new edge is added to the key exchange graph.)

These steps 1–4 are repeated until either exactly one player remains in the protocol or there are not enough cards left to complete step 2 even if two or more players remain. In the first case the key exchange graph becomes a tree.

In the second case the key exchange graph does not become a connected graph and hence does not become a tree.

Considering various procedures for choosing the proposer P_s in step 1, we obtain the class of *key set protocols*, where all the procedures are functions $\Gamma_k \to V$.

2.2 Malicious Adversary

If a key set protocol works for a signature γ, then the key exchange graph must become a tree for any deal $\mathcal{C}$ having the signature γ. Hence, whoever has the card y contained in the proposed key set $K = \{x, y\}$, the key exchange graph should become a tree. The *malicious adversary* determines who holds the card y. We use a function $\mathcal{A} : \Gamma_k \times V \to V \cup \{e\}$ to represent a malicious adversary, where e is Eve's index. The inputs to the function $\mathcal{A}(\gamma, s)$ are the current signature $\gamma \in \Gamma_k$ and the index $s \in V$ of a proposer P_s chosen by the protocol. Its output is either the index t of a player P_t remaining in the protocol or the index e of Eve; $\mathcal{A}(\gamma, s) = t \neq e$ means that player P_t holds card y; and $\mathcal{A}(\gamma, s) = e$ means that Eve holds card y.

From now on, we denote by $\gamma = (c_1, c_2, \cdots, c_k; c_e)$ the current signature, and denote by $\gamma'_{(s,\mathcal{A})} = (c'_1, c'_2, \cdots, c'_{k'}; c'_e)$ the resulting signature after executing steps 1–4 under the assumption that P_s proposes a key set $K = \{x, y\}$ and $y \in C_{\mathcal{A}(\gamma,s)}$. We sometimes write γ' instead of $\gamma'_{(s,\mathcal{A})}$ if it is clear from context.

Consider a signature $\gamma = (8, 7, 6, 4, 4, 4, 3, 2, 1; 3)$ as an example. Then, as illustrated in Fig. 1(a), the size of the hand of each player or Eve can be represented by white rectangles. For example, if the malicious adversary $\mathcal{A}$ satisfies $\mathcal{A}(\gamma, 2) = \mathcal{A}(\gamma, 3) = 1$, then $\gamma'_{(2,\mathcal{A})} = (7, 6, 4, 4, 4, 3, 2, 1; 3)$ as in Fig. 1(b), and $\gamma'_{(3,\mathcal{A})} = (7, 7, 4, 4, 4, 3, 2, 1; 3)$ as in Fig. 1(c). In Figs. 1(b) and (c), the shaded rectangles correspond to the discarded cards.

If an optimal key set protocol chooses a proposer P_s for $\gamma \in W$, then $\gamma'_{(s,\mathcal{A})} \in W$ for any malicious adversary $\mathcal{A}$; for convenience sake any signature $\gamma = (c_1; c_e)$ with $k = 1$ is assumed to be in W.

It follows from the definition of a key set protocol that if two players P_i and P_j hold hands of the same size, that is, $c_i = c_j$, then

$$\forall \mathcal{A} \ \ \gamma'_{(i,\mathcal{A})} \in W \iff \forall \mathcal{A} \ \ \gamma'_{(j,\mathcal{A})} \in W.$$

Hence, if there exist two or more players P_i with $c_i = c_s$ (including the proposer P_s), then one may assume without loss of generality that P_s has the largest index among all these players. We call it *Assumption 1 for convenience sake*. Similarly, if $\mathcal{A}(\gamma, s) = t \neq e$ and there exist two or more players P_i with $c_i = c_t$ and $i \neq s$ (including P_t), then one may assume without loss of generality that P_t has the largest index among all these players. We call it *Assumption 2 for convenience sake*. Under the two assumptions above, $\gamma'_{(s,\mathcal{A})} = (c'_1, c'_2, \cdots, c'_{k'}; c'_e)$ satisfies $c'_1 \geq c'_2 \geq \cdots \geq c'_{k'}$ since γ satisfies $c_1 \geq c_2 \geq \cdots \geq c_k$.

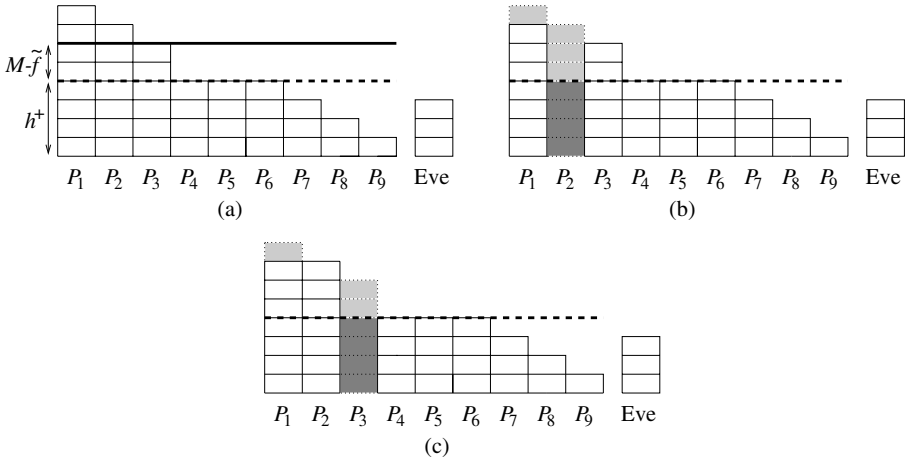


Fig. 1. The alteration of a signature.

2.3 Feasible Players

Fischer and Wright define a "feasible" player for a proposer as follows [3,6]. Let $k \geq 3$. If $c_e \geq 1$, P_i with $c_i = 1$ were chosen as a proposer, and $\mathcal{A}(\gamma, i) = e$, then P_i's hand would become empty although she remains in the protocol, and hence the key exchange graph would not become a tree. On the other hand, if $c_e = 0$, then $\mathcal{A}(\gamma, i) \neq e$ and hence the protocol appears to be able to choose P_i with $c_i = 1$ as a proposer; however, if $\mathcal{A}(\gamma, i) = j$ and $c_j = 1$, then P_j's hand would become empty and hence the key exchange graph would not become a tree. Thus the protocol can choose P_i with $c_i = 1$ as a proposer only if $c_e = 0$ and $c_j \geq 2$ for every j such that $1 \leq j \leq k$ and $j \neq i$, that is, only if $c_e = 0$, $i = k$ and $c_{k-1} \geq 2$. Remember that $c_1 \geq c_2 \geq \cdots \geq c_k$ is assumed. Hence, we say that a player P_i is *feasible* if the following condition (1) or (2) holds.

(1) $c_i \geq 2$.
(2) $c_e = 0$, $c_i = 1$ with $i = k$, and $c_{k-1} \geq 2$.

Thus, if the hands of all the players remaining in a protocol are not empty, i.e. $c_k \geq 1$, and the proposer P_s is feasible, then the hands of all the players remaining in the protocol will not be empty at the beginning of the succeeding execution of steps 1–4, i.e. $c'_{k'} \geq 1$. Note that there will not always exist a feasible player at the beginning of the succeeding execution of steps 1–4 even if the proposer P_s is feasible.

We define a mapping f from Γ_k to $\{0, 1, 2, \cdots, k\}$, as follows: $f(\gamma) = i$ if P_i is the feasible player with the smallest hand (ties are broken by selecting the player having the largest index); and $f(\gamma) = 0$ if there is no feasible player. For example, if $\gamma = (4, 3, 2, 2, 1, 1; 3)$, then $f(\gamma) = 4$. If $\gamma = (4, 4, 3, 3, 1; 0)$, then $f(\gamma) = k = 5$ because $c_e = 0$, $c_k = 1$ and $c_{k-1} \geq 2$. If $\gamma = (1, 1, 1; 2)$, then

$f(\gamma) = 0$ because there is no feasible player. Hereafter we often denote $f(\gamma)$ simply by f and $f(\gamma')$ by f'.

The following Lemma 1 immediately holds [3,10].

Lemma 1 ([3,10]) *The following* (a)–(d) *hold.*

(a) *If* $\gamma \in W$, *then* $c_k \geq 1$ [3].
(b) *If* $k \geq 3$ *and* $\gamma \in W$, *then* $f \geq 1$ [3].
(c) *If* $c_k \geq 1$, *then* $c_i = 1$ *for every* i *such that* $f + 1 \leq i \leq k$ [10].
(d) *If* $f \geq 1$ *and* $c_f = 1$, *then* $f = k$, $c_k = 1$, $c_{k-1} \geq 2$, $c_e = 0$, *and* $\gamma \in W$ [3].

2.4 SFP Protocol

Fischer and Wright give the *SFP (smallest feasible player) protocol* as a key set protocol [3,6]. The SFP protocol always chooses the feasible player with the smallest hand as a proposer, that is, chooses the proposer P_s as follows:

$$s = \begin{cases} f(\gamma) & if\ 1 \leq f(\gamma) \leq k; \\ 1 & if\ f(\gamma) = 0. \end{cases}$$

Fischer and Wright show that the SFP protocol is optimal [3,6].

Theorem 2 ([3,6]) *The SFP protocol is optimal.*

Not only the SFP protocol but also many other key set protocols are optimal. This paper provides a complete characterization of optimal key set protocols.

2.5 Necessary and Sufficient Condition for $\gamma \in W$

For $k = 2$, the following Theorem 3 provides a necessary and sufficient condition for $\gamma \in W$ [3].

Theorem 3 ([3]) *Let* $k = 2$. *Then* $\gamma \in W$ *if and only if* $c_2 \geq 1$ *and* $c_1 + c_2 \geq c_e + 2$.

For $k = 3$, the following Theorem 4 provides a necessary and sufficient condition for $\gamma \in W$ [10].

Theorem 4 ([10]) *Let* $k = 3$. *Then* $\gamma \in W$ *if and only if* $c_3 \geq 1$ *and* $c_1 + c_3 \geq c_e + 3$.

For $k \geq 4$, the following Theorem 5 provides a necessary and sufficient condition for $\gamma \in W$ [10]. Hereafter let $B = \{i \in V \mid c_i = 2\}$, and let $b = \lfloor |B|/2 \rfloor$.

Theorem 5 ([10]) *Let* $k \geq 4$, $c_k \geq 1$ *and* $f \geq 1$. *Then* $\gamma \in W$ *if and only if*

$$\sum_{i=1}^{k} \max\{c_i - h^+, 0\} \geq \widetilde{f}, \tag{1}$$

where

$$\bar{f} = f - \delta, \tag{2}$$

$$\widetilde{f} = \bar{f} - 2\epsilon, \tag{3}$$

$$h = c_e - c_k + k - \bar{f}, \tag{4}$$

$$h^+ = h + \epsilon, \tag{5}$$

$$\delta = \begin{cases} 0 \; if \; f = 1; \\ 1 \; if \; 2 \le f \le k-1; \\ 2 \; if \; f = k \; and \; c_{k-1} \ge c_k + 1; \; and \\ 3 \; if \; f = k \; and \; c_{k-1} = c_k, \end{cases} \tag{6}$$

and

$$\epsilon = \begin{cases} \max\{\min\{c_2 - h, b\}, 0\} & if \; 5 \le f \le k-1; \\ \max\{\min\{c_2 - h, b-1\}, 0\} & if \; 5 \le f = k \; and \; c_e \ge 1; \; and \\ 0 & otherwise. \end{cases} \tag{7}$$

For example, one can observe a signature $\gamma = (8, 7, 6, 4, 4, 4, 3, 2, 1; 3)$ (see Fig. 1(a)) satisfies Eq. (1) in Theorem 5 as follows. The signature γ satisfies $k = 9$ and $f = 8$. Thus by Eq. (6) $\delta = 1$. Since $B = \{8\}$, $b = 0$ and hence by Eq. (7) $\epsilon = 0$. Thus, by Eqs. (2) and (3) $\widetilde{f} = \bar{f} = 8 - 1 = 7$, and by Eqs. (4) and (5) $h^+ = h = 3 - 1 + 9 - 7 = 4$. Therefore,

$$\sum_{i=1}^{k} \max\{c_i - h^+, 0\} = 4 + 3 + 2 = 9 > 7 = \widetilde{f},$$

and hence the signature γ satisfies Eq. (1). (Note that $\sum_{i=1}^{k} \max\{c_i - h^+, 0\}$ is equal to the number of rectangles above the dotted line in Fig. 1(a).) Thus $\gamma \in W$.

Eq. (1) looks in appearance to be similar to the condition for a given degree sequence to be "graphical" [1,7,8,11].

Since $c_1 \ge c_2 \ge \cdots \ge c_k$ is assumed, Eq. (1) is equivalent to

$$\sum_{i=1}^{\widetilde{f}} \max\{c_i - h^+, 0\} \ge \widetilde{f} \tag{8}$$

where the summation is taken over all i, $1 \le i \le \widetilde{f}$, although the summation in Eq. (1) is taken over all i, $1 \le i \le k$ [10].

We define $\delta', \epsilon', b', \bar{f}', \widetilde{f}', h', h^{+\prime}$ and B' for γ' as we did for γ.

3 Main Results

In this section we give a complete characterization of optimal key set protocols. We first define some terms in Section 3.1, and then give the characterization in Section 3.2.

3.1 Definition of Selectable Players

In this subsection we define a "selectable" player that can be chosen as a proposer by an optimal key set protocol. We will give a complete characterization of "selectable" players in the succeeding subsection. The characterization immediately provides a complete characterization of optimal key set protocols.

The SFP protocol, which always chooses the feasible player P_f with the smallest hand, is optimal. However, a key set protocol which chooses an arbitrary feasible player is not necessarily optimal. We define a "selectable" player as follows.

Definition 6 *We say that a player P_i is* selectable *for γ if $\gamma'_{(i,\mathcal{A})} \in W$ for any malicious adversary $\mathcal{A}$.*

When $\gamma \in W$, the proposer chosen by an optimal key set protocol is a selectable player, of course. Since the SFP protocol is optimal, P_f is a selectable player if $\gamma \in W$.

Definition 6 implies that $\gamma \in W$ if and only if there exists at least one selectable player. In other words, $\gamma \in L$ if and only if there exists no selectable player.

Furthermore, a key set protocol is optimal if and only if the protocol always chooses a selectable player as a proposer whenever such a player exists. Thus, in the remainder of the paper, we characterize the set of all selectable players.

3.2 Characterization of Selectable Players

In this subsection we give a necessary and sufficient condition for a player to be selectable.

If $\gamma \in L$, then there is no selectable player. Therefore it suffices to obtain a necessary and sufficient condition for a player to be selectable only if $\gamma \in W$.

We first characterize the selectable players for $k = 2$ as in the following Theorem 7.

Theorem 7 *Let $k = 2$ and $\gamma \in W$. Then a player P_i is selectable if and only if $c_i \geq 2$ or $c_e = 0$.*

Proof. Let $k = 2$ and $\gamma \in W$. By Lemma 1(a) $c_2 \geq 1$.

We first prove the necessity. Suppose for a contradiction that $c_i = 1$ and $c_e \geq 1$ although P_i is selectable. Then one may assume that $i = 2$ by Assumption 1 for convenience sake when P_i is chosen as a proposer. Since $\gamma'_{(2,\mathcal{A})} = (c_1, 0; c_e - 1)$ for an adversary $\mathcal{A}$ such that $\mathcal{A}(\gamma, 2) = e$, we have $\gamma'_{(2,\mathcal{A})} \in L$ by Lemma 1(a). Thus P_2, i.e. P_i, is not selectable, a contradiction.

We next prove the sufficiency. Assume that $c_i \geq 2$ or $c_e = 0$. Then it suffices to show that $\gamma'_{(i,\mathcal{A})} \in W$ for any adversary $\mathcal{A}$. There are the following two cases to consider.

Case 1: $\mathcal{A}(\gamma, i) \neq e$.

In this case γ' satisfies $k' = 1$ and hence $\gamma' \in W$.

Case 2: $\mathcal{A}(\gamma, i) = e$.

In this case $c_e \geq 1$, and hence $c_i \geq 2$ because we assumed that $c_i \geq 2$ or $c_e = 0$. If $i = 1$ and $c_1 \geq c_2 + 1$, then $\gamma' = (c_1 - 1, c_2; c_e - 1)$; otherwise, $\gamma' = (c_1, c_2 - 1; c_e - 1)$. Thus, in either case, $c'_1 + c'_2 = (c_1 + c_2) - 1$ and $c'_e = c_e - 1$. On the other hand, since $\gamma \in W$, by Theorem 3 $c_1 + c_2 \geq c_e + 2$. Therefore $c'_1 + c'_2 \geq (c_e + 2) - 1 = c_e + 1 = c'_e + 2$. Furthermore, since $c_i \geq 2$, $c'_2 \geq 1$. Thus, by Theorem 3 $\gamma' \in W$. ∎

We next characterize the selectable players for $k = 3$. It has been known that, if $c_k \geq 1$ and $c_1 + c_k \geq c_e + k$, then any key set protocol choosing an arbitrary feasible player as a proposer works for γ [3,6]; thus the following Lemma 8 immediately holds.

Lemma 8 *Let $c_k \geq 1$ and $c_1 + c_k \geq c_e + k$. Then every player P_i such that $1 \leq i \leq f$ is selectable.*

Furthermore, it is obvious that any non-feasible player is not selectable when $k \geq 3$; thus we have the following Lemma 9.

Lemma 9 *Let $k \geq 3$. If a player P_i is selectable, then $1 \leq i \leq f$.*

By using Theorem 4, Lemmas 8 and 9, one can easily prove that the selectable players for $k = 3$ are characterized as in the following Theorem 10.

Theorem 10 *Let $k = 3$ and $\gamma \in W$. Then a player P_i is selectable if and only if $1 \leq i \leq f$.*

Proof. Let $k = 3$ and $\gamma \in W$. Then by Theorem 4 $c_3 \geq 1$ and $c_1 + c_3 \geq c_e + 3$. Thus Lemma 8 implies the sufficiency. Furthermore Lemma 9 implies the necessity. ∎

We finally characterize the selectable players for $k \geq 4$. Before giving the characterization, we first give some definitions.

In a key set protocol, for every $i, j \in V$ such that $i \neq j$ and $c_i = c_j$,

$$P_i \text{ is selectable} \iff P_j \text{ is selectable.}$$

Thus, if there exist two or more players holding hands of the same size, then it suffices to determine whether an arbitrary player among such players is selectable or not. For example, if $\gamma = (8, 7, 6, 4, 4, 4, 3, 2, 1; 3)$, then one can choose P_6 as a "representative" player among the three players P_4,P_5 and P_6 who have hands of size 4. As in this example, we choose the player with the largest index among all the players holding hands of the same size as a "representative" player, and determine whether the chosen "representative" player is selectable or not. Let V_r be the set of indices of all the "representative" players. That is,

$$V_r = \{i \in V \mid i = \max X \text{ and } X \in V/R\},$$

where V/R is the quotient set of V under the equivalence relation $R = \{(i,j) \in V \times V \mid c_i = c_j\}$. For example, $V_r = \{1,2,3,6,7,8,9\}$ for the above signature γ. It suffices to give a necessary and sufficient condition for a player P_i, $i \in V_r$, to be selectable. Of course, such a necessary and sufficient condition immediately yields a complete characterization of all selectable players (whose indices are not necessarily in V_r).

Let P_{f_m} be the player who holds the hand of the same size as P_f and has the smallest index, that is,

$$f_m = \min\{i \in V \mid c_i = c_f\}.$$

From now on we define

$$M = \sum_{j=1}^{k} \max\{c_j - h^+, 0\}.$$

Note that M is the same as the left side of Eq. (1) in Theorem 5. We define M' for γ' as we did for γ.

Define $\bar{\epsilon}$ by the following Eq. (9), which is obtained by replacing c_2 with c_3 in Eq. (7):

$$\bar{\epsilon} = \begin{cases} \max\{\min\{c_3 - h, b\}, 0\} & \textit{if } 5 \le f \le k-1; \\ \max\{\min\{c_3 - h, b-1\}, 0\} & \textit{if } 5 \le f = k \textit{ and } c_e \ge 1; \textit{ and} \\ 0 & \textit{otherwise.} \end{cases} \tag{9}$$

Since $c_3 \le c_2$, Eqs. (7) and (9) imply

$$0 \le \bar{\epsilon} \le \epsilon. \tag{10}$$

Furthermore, define *Conditions 1* and *2* as follows.

(Condition 1)

$5 \le f = k$ and $c_{k-2} = c_{k-1} = c_k + 1$.

(Condition 2)

$c_{f_m-2} = c_{f_m-1} = 3$, $|B|$ is an odd number, and the following (i) or (ii) holds.

(i) $6 \le f \le k-1$ and $c_2 - h \ge b + 1$.

(ii) $6 \le f = k$, $c_e \ge 1$, $b \ge 1$ and $c_2 - h \ge b$.

Define λ as follows:

$$\lambda = \begin{cases} 2 \textit{ if Condition 1 holds}; \\ 3 \textit{ if Condition 2 holds}; \textit{ and} \\ 0 \textit{ otherwise.} \end{cases} \tag{11}$$

Finally, define $\widetilde{\epsilon}$ as follows:

$$\widetilde{\epsilon} = \begin{cases} \max\{\min\{c_2 - h - 1, b-1\}, 0\} & \textit{if } f \ge 8,\ c_k = 1 \textit{ and } \lambda = 2; \textit{ and} \\ 0 & \textit{otherwise.} \end{cases} \tag{12}$$

We are now ready to give a complete characterization of the selectable players for $k \ge 4$ as in the following Theorem 11.

Theorem 11 *Let $k \geq 4$ and $\gamma \in W$. Then a player P_i such that $i \in V_r$ is selectable if and only if $1 \leq i \leq f$ and*

$$\begin{cases} c_2 - h^+ \leq M - \widetilde{f} - (\epsilon - \bar{\epsilon}) & \text{if } i \leq 2; \\ \sum_{j=1}^{\widetilde{f}-\lambda-2\widetilde{\epsilon}} \max\{c_j - (h^+ + \widetilde{\epsilon} + 1), 0\} \geq \widetilde{f} - \lambda - 2\widetilde{\epsilon} & \\ \qquad \text{if } i = f_m - 1 \geq 4 \text{ and } \lambda \neq 0; \text{ and} & \\ c_i - h^+ \leq M - \widetilde{f} & \text{otherwise.} \end{cases} \tag{13}$$

If (i) $i \leq 2$ and $\epsilon - \bar{\epsilon} = 0$, (ii) $i = 3$, or (iii) $i \geq 4$ and $i \neq f_m - 1$ or $\lambda = 0$, then Eq. (13) in Theorem 11 becomes

$$\begin{cases} c_2 - h^+ \leq M - \widetilde{f} \text{ if } i \leq 2; \text{ and} \\ c_i - h^+ \leq M - \widetilde{f} \text{ if } i \geq 3. \end{cases} \tag{14}$$

Note that the most of signatures satisfy $\epsilon - \bar{\epsilon} = \lambda = 0$ and that very few signatures satisfy $\epsilon - \bar{\epsilon} \geq 1$ or $\lambda \neq 0$.

Consider the signature $\gamma = (8, 7, 6, 4, 4, 4, 3, 2, 1; 3)$ as an example again (see Fig. 1(a)). The signature γ satisfies $\epsilon = 0$ as mentioned in Section 2.5, and hence by Eq. (10) $\epsilon - \bar{\epsilon} = 0$. The signature γ does not satisfy Condition 1. Furthermore, since $f = f_m = 8$, we have $c_{f_m - 2} = 4 \neq 3$ and hence Condition 2 does not hold. Therefore, by Eq. (11) $\lambda = 0$. In addition, since the signature γ satisfies $f = 8$, $M = 9$, $\widetilde{f} = 7$ and $h^+ = 4$ as mentioned in Section 2.5, we have $M - \widetilde{f} = 2$. Therefore, Eq. (13) in Theorem 11, i.e. Eq. (14), implies that all the selectable players are the six players P_3,P_4,P_5,P_6,P_7 and P_8. These six players are the feasible players holding the hands whose sizes do not exceed the solid line in Fig. 1(a).

We now intuitively explain the correctness of Theorem 11. For simplicity, let $\epsilon - \bar{\epsilon} = \lambda = 0$, and consider a player P_i such that $i \geq 2$. Theorem 5 implies that a necessary and sufficient condition for $\gamma \in W$ is that $M \geq \widetilde{f}$, i.e. there are $\widetilde{f}$ or more rectangles above the dotted line in Fig. 1(a). Thus, a signature $\gamma \in W$ has $M - \widetilde{f}$ "spare" rectangles. That is, even if one removes at most $M - \widetilde{f}$ rectangles above the dotted line, γ still remains in W, but if one removed $(M - \widetilde{f}) + 1$ or more rectangles above the dotted line, then γ would be in L. Further, in order for a player P_i to be selectable, there must exist at least $\widetilde{f}'$ rectangles above the dotted line in the figure of $\gamma'_{(i,\mathcal{A})}$ (e.g. Fig. 1(b) or (c)) for any malicious adversary $\mathcal{A}$. For some adversary $\mathcal{A}$, the number of the rectangles above the dotted line decreases by $1 + (c_i - h^+)$ when the proposer is P_i, as one can immediately observe from Fig. 1(b) or (c). Note that these $1 + (c_i - h^+)$ rectangles are lightly shaded in Figs. 1(b) and (c). Furthermore, since the number of the feasible players decreases by exactly one, we have $\widetilde{f}' = \widetilde{f} - 1$. Hence, if $c_i - h^+$ were greater than the number $M - \widetilde{f}$ of the "spare" rectangles, then $M' = M - \{1 + (c_i - h^+)\} < \widetilde{f} - 1 = \widetilde{f}'$ and hence γ' would be in L. Therefore, a player P_i such that $c_i - h^+ > M - \widetilde{f}$ is not selectable. On the other hand, if $c_i - h^+ \leq M - \widetilde{f}$, then γ' will still remain in W, and hence a player P_i such that $c_i - h^+ \leq M - \widetilde{f}$ is selectable. This is the intuitive reason why Theorem 11 holds.

Due to the page limitation, we cannot include a proof of Theorem 11 in this extended abstract; see [9].

4 Conclusion

A key set protocol is determined by giving a procedure for choosing a proposer. In this paper, we defined a player to be selectable if the player can be chosen as a proposer by an optimal key set protocol, and gave a complete characterization of such selectable players in Theorem 11. Thus we succeeded in characterizing the set of *all* optimal key set protocols.

Using Theorem 11, one can efficiently find all selectable players in time $O(k)$. Let P_j be the selectable player having the smallest index j. Then one may intuitively expect that all players P_i such that $j \leq i \leq f$ are selectable. However, it is surprisingly not the case. Theorem 11 implies that all the players such that $j \leq i \leq f$ and $c_i \neq c_{f_m-1}$ are selectable but P_{f_m-1} may or may not be selectable. Consider a signature $\gamma = (5, 5, 5, 4, 4, 3, 3, 2, 1; 2)$ as an example. Then γ satisfies $f = 8$, $f_m - 1 = 7$, $\lambda = 3$, $h^+ = 3$, $M = 8$, $\epsilon = \bar{\epsilon} = \widetilde{\epsilon} = 0$ and $\widetilde{f} = 7$. Thus Eq. (13) in Theorem 11 becomes

$$\begin{cases} c_2 - 3 \leq 1 & \textit{if } i \leq 2; \\ \sum_{\ell=1}^{4} \max\{c_\ell - 4, 0\} \geq 4 & \textit{if } i = 7; \textit{ and} \\ c_i - 3 \leq 1 & \textit{otherwise.} \end{cases}$$

Therefore, P_7 is not selectable, and all the selectable players are the three players P_4,P_5 and P_8. As in this example, the indices of selectable players are not necessarily consecutive numbers.

Using the characterization of selectable players, one can design many optimal key set protocols. Assume that $c_1 = c_2 = \cdots = c_k$ and $\gamma \in W$. Then in most cases the SFP protocol forms a spanning path of length k, i.e. a tree of radius $\lfloor k/2 \rfloor$, as the key exchange graph. On the other hand, using various optimal key set protocols, one can produce trees of various shapes as a key exchange graph, some of which would be appropriate for efficient broadcast of a secret message. For example, consider an optimal key set protocol which always chooses, as a proposer, the selectable player holding the largest hand; such a protocol forms a tree of a smaller radius than $\lfloor k/2 \rfloor$. Furthermore, we can choose the selectable player having the largest degree as a proposer and modify step 3 of the key set protocol in a way that either P_s or P_t who has the smaller degree drops out of the protocol whenever the resulting signature remains in W; such a protocol forms a tree of much smaller radius, especially when $c_1 = c_2 = \cdots = c_k$ is large. We have verified these facts by extensive computer simulation.

This paper addresses only the key set protocol, which establishes a one-bit secret key. On the other hand, the "transformation protocol" given by Fischer and Wright [4] establishes an n-bit secret key. For a signature $\gamma = (3, 2; 4) \in L$, any key set protocol does not work for γ, but the transformation protocol always establishes a one-bit secret key for γ. However, for a signature $\gamma = (4, 4, 4, 4; 4) \in$

W, any optimal key set protocol works for γ, but the transformation protocol cannot establish a one-bit secret key for γ. Thus a protocol entirely superior to the key set protocol has not been known.

References

1. T. Asano, "An $O(n \log \log n)$ time algorithm for constructing a graph of maximum connectivity with prescribed degrees," J. Comput. and Syst. Sci., vol. 51, pp. 503–510, 1995.
2. M. J. Fischer, M. S. Paterson, and C. Rackoff, "Secret bit transmission using a random deal of cards," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, vol. 2, pp. 173–181, 1991.
3. M. J. Fischer and R. N. Wright, "An application of game-theoretic techniques to cryptography," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, vol. 13, pp. 99–118, 1993.
4. M. J. Fischer and R. N. Wright, "An efficient protocol for unconditionally secure secret key exchange," Proc. of the 4th Annual Symposium on Discrete Algorithms, pp. 475–483, 1993.
5. M. J. Fischer and R. N. Wright, "Bounds on secret key exchange using a random deal of cards," J. Cryptology, vol. 9, pp. 71–99, 1996.
6. M. J. Fischer and R. N. Wright, "Multiparty secret key exchange using a random deal of cards," Proc. Crypto '91, Lecture Notes in Computer Science, Springer-Verlag, vol. 576, pp. 141–155, 1992.
7. S. L. Hakimi, "On realizability of a set of integers as degrees of the vertices of a linear graph. I," J. SIAM Appl. Math., vol. 10, no. 3, pp. 496–506, 1962.
8. F. Harary, "Graph Theory," Addison-Wesley, Reading, Mass., 1969.
9. T. Mizuki, "Sharing unconditionally secure secret keys," Ph.D. Thesis, Tohoku University, Sendai, 2000.
10. T. Mizuki, H. Shizuya, and T. Nishizeki, "Dealing necessary and sufficient numbers of cards for sharing a one-bit secret key," Proc. Eurocrypt '99, Lecture Notes in Computer Science, Springer-Verlag, vol. 1592, pp. 389–401, 1999.
11. E. F. Schmeichel and S. L. Hakimi, "On planar graphical degree sequences," SIAM J. Appl. Math., vol. 32, no. 3, pp. 598–609, 1977.

On the Complexity of Integer Programming in the Blum-Shub-Smale Computational Model

Valentin E. Brimkov[1] and Stefan S. Dantchev[2]

[1] Eastern Mediterranean University,
Famagusta, P.O. Box 95, TRNC, Via Mersin 10, Turkey
brimkov.as@mozart.emu.edu.tr
[2] BRICS - Basic Research in Computer Science,
Centre of the Danish National Research Foundation,
University of Aarhus, Department of Computer Science,
Ny Munkegade, Bldg. 540, DK-8000 Aarhus C., Denmark
dantchev@brics.dk

Abstract. In the framework of the Blum-Shub-Smale real number model [3], we study the *algebraic complexity* of the integer linear programming problem ($\mathrm{ILP}_{\mathbf{R}}$) : Given a matrix $A \in \mathbf{R}^{m\times n}$ and vectors $b \in \mathbf{R}^m$, $d \in \mathbf{R}^n$, decide whether there is $x \in \mathbf{Z}^n$ such that $Ax \leq b$, where $\mathbf{0} \leq x \leq d$. The main contributions of the paper are the following:
- An $O(m \log ||d||)$ algorithm for $\mathrm{ILP}_{\mathbf{R}}$, when the value of n is fixed. As a corollary, we obtain under the same restriction a tight algebraic complexity bound $\Theta\left(\log \frac{1}{a_{\min}}\right)$, $a_{\min} = \min\{a_1, \ldots, a_n\}$, for the knapsack problem ($\mathrm{KP}_{\mathbf{R}}$) : Given $a \in \mathbf{R}^n_+$, decide whether there is $x \in \mathbf{Z}^n$ such that $a^T x = 1$. We achieve these results in particular through a careful analysis of the algebraic complexity of the Lovász' basis reduction algorithm and the Kannan-Bachem's Hermite normal form algorithm, which may be of interest in its own.
- An $O\left(mn^5 \log n\,(n + \log ||d||)\right)$ depth *algebraic decision tree* for $\mathrm{ILP}_{\mathbf{R}}$, for every m and n.
- A new lower bound for 0/1 $\mathrm{KP}_{\mathbf{R}}$. More precisely, no algorithm can solve 0/1 $\mathrm{KP}_{\mathbf{R}}$ in $o(n \log n)\, f(a_1, \ldots, a_n)$ time, even if f is an *arbitrary continuous* function of n variables. This result appears as an alternative to the well-known Ben-Or's bound $\Omega(n^2)$ [1] and is independent upon it.
Keywords: *Algebraic complexity, Complexity bounds, Integer programming, Knapsack problem*

1 Introduction

We study the *algebraic complexity* of the following integer linear programming (ILP) problem:

($\mathrm{ILP}_{\mathbf{R}}$) Given a matrix $A \in \mathbf{R}^{m\times n}$ and vectors $b \in \mathbf{R}^m, d \in \mathbf{R}^n$, decide whether there is $x \in \mathbf{Z}^n$ such that $Ax \leq b$, where $\mathbf{0} \leq x \leq d$.

The input entries are arbitrary *real* numbers and, accordingly, the adopted model of computation is a *real number model*. This kind of model has been traditionally

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 286–300, 2000.
© Springer-Verlag Berlin Heidelberg 2000

used in scientific computing, computational geometry, and (although not explicitly) numerical analysis (see, e.g., [18,19,22]). In our study we conform mainly to the model presented in [3], known as the BSS-model (named after its creators Blum, Shub and Smale). In the BSS-model, the assumption is that all the real numbers in the input have unit size, and the basic algebraic operations $+, -, *, /$ and the relation $\leq$ are executable at unit cost. Thus the algebraic complexity of a computation on a problem instance is the number of operations and branchings performed to solve the instance. For more details on the BSS-model and complexity theory over arbitrary rings, we refer to [3]. We notice that in this new theory, aimed at providing a complexity framework for disciplines like those mentioned above, an important issue is seen in the comparison of results over the reals with classical results over the integers, which may help elucidate some fundamental concepts, like computability and complexity.

At this point it is important to mention that the requirement for bounded domain (i.e., $\mathbf{0} \leq x \leq d$) is essential and dictated by the very nature of the problem, namely by the fact that the coefficients may be *irrational* numbers. In such a case, a problem with unbounded domain may be, in general, *undecidable*, as shown in [4].

In a classical setting, integer linear programming with integer or rational inputs is among the best-studied combinatorial problems. A substantial body of literature, impossible to report here, has been developed on the subject. In particular, it is well-known that ILP is NP-complete [8]. Comparatively less is known about the complexity of $\mathrm{ILP}_{\mathbf{R}}$ in the framework of the BSS-model. Some related results are reported in [3,1,17,7,4]. In [2] Blum et al. pose the problem of studying the complexity of an important special case of $\mathrm{ILP}_{\mathbf{R}}$, known as the "real" knapsack problem:

$(\mathrm{KP}_{\mathbf{R}})$ Given $a \in \mathbf{R}^n_+$, decide if there is $x \in \mathbf{Z}^n$ such that $a^T x = 1$.

With the present paper we take a step towards determining $\mathrm{ILP}_{\mathbf{R}}$'s and $\mathrm{KP}_{\mathbf{R}}$'s complexity. Our main contributions are the following.

1. An $O(m \log ||d||)$ algorithm for $\mathrm{ILP}_{\mathbf{R}}$ when the value of n is fixed (Section 2).
   A similar result is known for the integer case, namely, the well-known Lenstra's algorithm for ILP of a fixed dimension n [12]. Our algorithm consists of two stages: a reduction of the given real input to an integer input determining the same admissible set, followed by an application of Lenstra's algorithm. The first stage involves simultaneous Diophantine approximation techniques, while the second employs two well-known algorithms: the Lovász' basis reduction algorithm [13] and the Kannan-Bachem's Hermite normal form algorithm [10]. It is straightforward to obtain an upper time complexity bound that is *quadratic* in $\log ||d||$. Our more detailed analysis reveals that the actual complexity of the latter two algorithms (and, as a consequence, of the entire algorithm) is *linear* in $\log ||d||$.

Applied to the knapsack problem $\mathrm{KP}_{\mathbf{R}}$ of fixed dimension n, our algorithm has complexity $O\left(\log \frac{1}{a_{\min}}\right)$, $a_{\min} = \min\{a_1, \ldots, a_n\}$, and turns out to be *optimal*.
In view of the fact that the Lovász' basis reduction algorithm and the Kannan-Bachem's Hermite normal form algorithm are fundamental and very important combinatorial algorithms, we believe that their algebraic complexity analysis within the BSS-model may be of interest in its own.

2. An $O\left(mn^5 \log n\,(n + \log ||d||)\right)$ depth *algebraic decision tree* for $\mathrm{ILP}_{\mathbf{R}}$, for every m and n (i.e., in a model which is nonuniform with respect to them) (Section 3).
This result is in the spirit of the well-known Meyer auf der Heide's $n^4 \log n + O\left(n^3\right)$ depth *linear decision tree* for 0/1 $\mathrm{KP}_{\mathbf{R}}$ (i.e., $\mathrm{KP}_{\mathbf{R}}$ with $x \in \{0,1\}^n$) [14].
3. A new lower bound for 0/1 $\mathrm{KP}_{\mathbf{R}}$. More precisely, no algorithm can solve 0/1 $\mathrm{KP}_{\mathbf{R}}$ in $o\left(n \log n\right) f\left(a_1, \ldots, a_n\right)$ time, even if f is an *arbitrary continuous* function of n variables (Section 4).
This result appears as an alternative to the well-known Ben-Or's bound $\Omega(n^2)$ [1] and is independent upon it, in the sense that neither of both results is superior to or implies the other.

2 Analysis of the Basic Algorithms

In this section, we analyze the Lovász lattice basis reduction algorithm [13] and the Kannan and Bachem's Hermite normal form algorithm [10]. It is well-known that these are polynomial within the classical computational model. This implies that, within the BSS model, they are polynomial with respect to the dimensions m and n of the input matrices and the maximal bit-size S of their integer (or rational) entries. Our deeper analysis shows that they are *linear* in S.

2.1 Some Useful Facts

In this section, we state some simple facts about vectors and matrices with rational entries of bit-size at most S. Although trivial, these facts will be instrumental in analyzing the algorithms in the next sections.

1. Let a be a non-zero rational number. Then $1/2^S \leq |a| \leq 2^S$.
2. Let b_1, b_2 be non-orthogonal n-dimensional rational vectors. Then $1/2^{2nS} \leq |\langle b_1, b_2 \rangle| \leq n2^{2S}$.
3. Let B be a non-singular $n \times n$ rational matrix. Then $1 \Big/ 2^{n^2 S} \leq |\det(B)| \leq n!2^{nS}$.
4. Let B_i be an $n \times i$ rational matrix of rank i, $i \leq n$. Then $1 \Big/ 2^{2n^2 S} \leq \left|\det\left(B^T B\right)\right| \leq n!2^{n(\log n + 2S)}$.

2.2 Lovász Lattice Basis Reduction Algorithm

In the description and analysis of the algorithm we follow [9]. The input consists of linearly independent vectors $b_1, b_2, \ldots b_n \in \mathbf{Q}^n$, considered as a basis for a lattice L. The algorithm transforms them iteratively. At the end, they form a basis for L which is reduced in the Lovázs sense.

First we recall some definitions, then describe the Lovász lattice basis reduction algorithm, itself. With a basis $b_1, b_2, \ldots b_n$, we associate the orthogonal system $b_1^*, b_2^*, \ldots b_n^*$, where b_i^* is the component of b_i which is orthogonal to $b_1, b_2, \ldots b_{i-1}$. The vectors $b_1^*, b_2^*, \ldots b_n^*$ can be computed by Gram-Schmidt orthogonalization:

$b_1^* = b_1$,
$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*, \ \ 2 \leq i \leq n$,
where $\mu_{i,j} = \langle b_i, b_j^* \rangle \Big/ \left\| b_j^* \right\|^2$.

The basis $b_1, b_2, \ldots b_n$ is *size-reduced* if all $|\mu_{i,j}| \leq \frac{1}{2}$. Given an arbitrary basis $b_1, b_2, \ldots b_n$, we can transform it into a size-reduced basis with the same Gram-Schmidt orthogonal system, as follows:

For every i from 2 to n; For every j from $i-1$ to 1;
Set $b_i := b_i - \lceil \mu_{i,j} \rfloor \, b_j$ and update $\mu_{i,k}$ for $1 \leq k \leq i-1$, by setting $\mu_{i,k} = \mu_{i,k} - \lceil \mu_{i,j} \rfloor \, \mu_{j,k}$.

Now, we can describe a variant of the Lovász lattice basis reduction algorithm.

1. *Initiation.* Compute the Gram-Schmidt quantities $\mu_{i,j}$ and b_i^* for $1 \leq j < i \leq n$. Size-reduce the basis.
2. *Termination condition.* If $\|b_i^*\|^2 \leq 2 \left\| b_{i+1}^* \right\|^2$ for $1 \leq i \leq n-1$, then stop.
3. *Exchange step.* Choose the smallest i such that $\|b_i^*\|^2 > 2 \left\| b_{i+1}^* \right\|^2$. Exchange b_i and b_{i+1}. Update the Gram-Schmidt quantities. Size-reduce the basis. Go to 2.

For completeness, we give formulae for updating the Gram-Schmidt quantities in step 3:

$$\|b_i^*\|_{new}^2 = \left\| b_{i+1}^* \right\|^2 + \mu_{i+1,i}^2 \|b_i^*\|^2$$
$$\left\| b_{i+1}^* \right\|_{new}^2 = \|b_i^*\|^2 \left\| b_{i+1}^* \right\|^2 \Big/ \|b_i^*\|_{new}^2$$
$$\mu_{i+1,i}^{new} = \mu_{i+1,i} \|b_i^*\|^2 \Big/ \|b_i^*\|_{new}^2$$
$$\begin{pmatrix} \mu_{i,j}^{new} \\ \mu_{i+1,j}^{new} \end{pmatrix} = \begin{pmatrix} \mu_{i+1,j} \\ \mu_{i,j} \end{pmatrix} \text{ for } 1 \leq j \leq i-1$$
$$\begin{pmatrix} \mu_{j,i}^{new} \\ \mu_{j,i+1}^{new} \end{pmatrix} = \begin{pmatrix} 1 & \mu_{i+1,i}^{new} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & -\mu_{i+1,i} \end{pmatrix} \begin{pmatrix} \mu_{j,i} \\ \mu_{j,i+1} \end{pmatrix} \text{ for } i+2 \leq j \leq n.$$

The other $\|b_i^*\|^2$'s and $\mu_{i,j}$'s do not change.

After termination of the above algorithm, we have a size-reduced basis for which $\|b_i^*\|^2 \leq 2 \left\| b_{i+1}^* \right\|^2$, $1 \leq i \leq n-1$. We call such a basis *reduced in the Lovász sense.* (There are other definitions of this concept in the literature, but

for our purposes they are essentially equivalent.) Important properties of such a basis are

$$\|b_1\| \leq 2^{\frac{n-1}{2}} \|(shortest\ vector\ in\ L)\|,$$

and

$$\prod_{i=1}^{n} \|b_i\| \leq 2^{\frac{n(n-1)}{4}} \det(L). \tag{1}$$

Let us analyze the running time of steps 2 and 3. Consider the function

$$F(b_1^*, b_2^*, \ldots b_n^*) := \prod_{i=1}^{n} \|b_i^*\|^{2(n-i)} = \prod_{i=1}^{n-1} det\left(B_i^T B_i\right),$$

where B_i is a matrix having $b_1, b_2, \ldots b_i$ as column vectors. No size-reduction operation changes F, as it does not change the $\|b_i^*\|$'s. After an exchange step, we obtain

$$\frac{F_{new}}{F} = \frac{\|b_i^*\|_{new}^2}{\|b_i^*\|^2} = \frac{\left\|b_{i+1}^*\right\|^2 + \mu_{i+1,i}^2 \|b_i^*\|^2}{\|b_i^*\|^2} < \frac{3}{4}. \tag{2}$$

It is not hard to see that every iteration of steps (2-3) consists of $O\left(n^2\right)$ basic arithmetic operations (because of the size-reduction, an updating needs only $O(n)$ such operations). The only problem might be the rounding operations $\lceil . \rfloor$ performed during the size-reductions. We observe that the absolute values of their arguments are at most $O\left(n\mu_{i+1,i}^{new}\right)$. Then the time needed for one such an operation is

$$O\left(\log n + \log \mu_{i+1,i}^{new}\right) = O\left(\log n + \log\left(\|b_i^*\|^2 \Big/ \|b_i^*\|_{new}^2\right)\right).$$

Thus, the time complexity of one iteration is

$$O\left(n^2\left(\log n + \log\left(\|b_i^*\|^2 \Big/ \|b_i^*\|_{new}^2\right)\right)\right) = O\left(n^2\left(\log n + \log \frac{F}{F_{new}}\right)\right).$$

Then the time complexity of all iterations is

$$O\left((\#iterations)\, n^2 \log n + n^2 \log \frac{F_{start}}{F_{end}}\right). \tag{3}$$

Because of (2), the number of iterations is $O\left(\log \frac{F_{start}}{F_{end}}\right)$, so that the overall complexity of steps 2 and 3 is

$$O\left(n^2 \log n \log \frac{F_{start}}{F_{end}}\right). \tag{4}$$

What remains is to estimate the running time of step 1 and the ratio $\frac{F_{start}}{F_{end}}$.

By definition, $\mu_{i,j}$ ($1 \leq j \leq i-1$) can be considered as a solution of the following linear system

$$\begin{pmatrix} \langle b_1, b_1 \rangle & & \langle b_1, b_{i-1} \rangle \\ & \ddots & \\ \langle b_{i-1}, b_1 \rangle & & \langle b_{i-1}, b_{i-1} \rangle \end{pmatrix} \begin{pmatrix} \mu_{i,1} \\ \vdots \\ \mu_{i,i-1} \end{pmatrix} = \begin{pmatrix} \langle b_1, b_i \rangle \\ \vdots \\ \langle b_{i-1}, b_i \rangle \end{pmatrix},$$

for $2 \leq i \leq n$. From here and fact 4 from Section 2.1, it is not difficult to deduce that before the size-reduction phase of step 1, $\|\mu_{i,j}\| \leq 2^{O\left(Sn^2\right)}$. The size-reduction itself takes $O\left(n^2\right)$ $\lceil . \rfloor$ operations on these numbers, so that the time complexity of step 1 is clearly $O\left(Sn^4\right)$.

Lastly, we need to estimate F_{start} and F_{end}. F_{start} is a product of the determinants of $n-1$ matrices $B_i^T B_i$ where $1 \leq i \leq n-1$, so that

$$|F_{start}| \leq 2^{O\left(n^2(\log n + S)\right)}.$$

To estimate F_{end}, let us observe that any of the vectors $b_1^{end}, b_2^{end}, \ldots b_n^{end}$ is an *integer* linear combination of $b_1^{start}, b_2^{start}, \ldots b_n^{start}$. Therefore, for $1 \leq i \leq n$, we have $B_i^{end} = B_n^{start} A_i$, where A_i is an $n \times i$ integer matrix. This implies

$$\det\left(B_i^{end^T} B_i^{end}\right) = \det\left(A_i^T A_i\right) \det\left(B_i^{start^T} B_i^{start}\right) \geq 1 \Big/ 2^{2n^2 S},$$

by fact 4 from Section 2.1. Consequently, $F_{end} \geq 1 \Big/ 2^{O\left(n^3 S\right)}$. Thus we obtain $\log \frac{F_{start}}{F_{end}} = O\left(n^3 S\right)$. Hence, the overall complexity of the Lovász basis reduction algorithm is $O\left(Sn^5 \log n\right)$.

Finally, we will prove that the bit-size of the entries of the reduced basis is $O\left(Sn^3\right)$. Let us recall inequality (1):

$$\prod_{i=1}^{n} \left\|b_i^{end}\right\| \leq 2^{\frac{n(n-1)}{4}} \det(L) = 2^{\frac{n(n-1)}{4}} \left|\det\left(B_n^{start}\right)\right|,$$

and denote with a the least common multiple of all entries of B_n^{start}. Note that the bit-size of a is $O\left(Sn^2\right)$. Since b_i^{end}'s are integer linear combinations of b_i^{start}'s, the vectors ab_i^{end} are integer. Therefore, we have

$$\prod_{i=1}^{n} \left\|ab_i^{end}\right\| \leq 2^{\frac{n(n-1)}{4}} a^n \left|\det\left(B_n^{start}\right)\right| \leq 2^{\frac{n(n-1)}{4}} 2^{Sn^3} n! 2^S = 2^{O\left(Sn^3\right)}.$$

Thus, every entry of aB_n^{end} is of bit-size $O\left(Sn^3\right)$ and so is every entry of B_n^{end}. In terms of the adopted denotations, we have proved the following lemma.

Lemma 1. *The algebraic complexity of Lovász' basis reduction algorithm is $O(Sn^5 \log n)$, and the bit-size of the entries in the reduced basis is $O(Sn^3)$.*

2.3 Kannan and Bachem's Hermite Normal Form Algorithm

In our description we follow [21]. The input for the algorithm is an $m \times n$ ($m \leq n$) integer matrix A of full rank. The algorithm uses the matrix

$$A' = \left(A \left| \begin{array}{ccc} M & & \\ & \ddots & \\ & & M \end{array} \right. \right),$$

where M is the absolute value of some nonsingular $m \times m$ minor of A. A' has the same Hermite normal form as A. The algorithm consists of the following five steps:

1. Cause all the entries of the matrix A to fall into the interval $[0, M)$, by adding to the first n columns of A' proper integer multiples of the last n columns;
2. For k from 1 to m do 3-4;
3. If there are $i \neq j$, $k \leq i, j \leq n+k$, such that $a'_{k,i} \geq a'_{k,j} > 0$, then subtract from the ith column the jth one multiplied by $\left\lfloor \frac{a'_{k,i}}{a'_{k,j}} \right\rfloor$. Then reduce the ith column modulo M. Go to 3;
4. Exchange the kth column and the only column with $a'_{k,i} > 0$;
5. For every i from 2 to n; for every j from 1 to $i-1$, add an integer multiple of the ith column to the jth one, to get $a'_{i,i} > a'_{i,j} \geq 0$.

In order to show that the time complexity is polynomial in m, n and linear in S, we need to analyze step 3. For this, we introduce the function

$$F\left(a'_{k,k}, a'_{k,k+1}, \ldots a'_{k,n+k}\right) := \prod_{\substack{k \leq i \leq n+k \\ a'_{k,i} > 0}} a'_{k,i}.$$

After one iteration of step 3, we have

$$\frac{F_{new}}{F} = \frac{a'_{k,i} - \left\lfloor \frac{a'_{k,i}}{a'_{k,j}} \right\rfloor a'_{k,j}}{a'_{k,i}},$$

which implies both $\frac{F_{new}}{F} < \frac{1}{2}$ and $\frac{F_{new}}{F} < \frac{a'_{k,j}}{a'_{k,i}}$. It is not hard to see that one iteration of step 3 can be performed in $O\left(m \log \frac{a'_{k,i}}{a'_{k,j}}\right) = O\left(m \log \frac{F}{F_{new}}\right)$ time. So, step 3 takes $O\left(m \log \frac{F_{start}}{F_{end}}\right)$ time. Since $F_{start} < M^{n+1}$, $F_{end} \geq 1$, and $M = O\left(m! 2^{mS}\right)$ by fact 3 of Section 2.1, the overall running time of step 3 is $O\left(nm\left(\log m + S\right)\right)$. Then the complexity of the Kannan-Bachem's algorithm is $O\left(nm^2\left(\log m + S\right)\right)$.

Since all the resulting integers are smaller than M, their bit-size is $O\left(Smn\right)$. Thus we have proved the following lemma.

Lemma 2. *Let A be an $m \times n$ ($m \leq n$) integer matrix of full rank. Then the algebraic complexity of the Kannan-Bachem's algorithm that reduces A into its Hermite normal form, is $O(m^2 n(\log m + S))$.*

3 Basic Results about $\mathrm{ILP}_{\mathbf{R}}$

In this section we use the analysis of the algorithms from the previous section to obtain the first two of the results announced in the Introduction.

To solve $\mathrm{ILP}_{\mathbf{R}}$ algorithmically within the BSS-model, we follow the idea of our method developed in [4]. There its complexity was analyzed within a strengthened version of the BSS-model, in which the floor operation $\lfloor . \rfloor$ is considered as a basic one, executable at unit cost. Here we will apply and analyze it within the standard BSS-model.

The algorithm employs in one of its stages the well-known algorithm for finding a simultaneous Diophantine approximation to a given rational vector. In particular, we will use the following lemma.

Lemma 3. *(see, e.g., [21, Corollary 6.4c]) There exists a polynomial algorithm which, given a vector $a \in \mathbf{Q}^n$ and a rational number ε, $0 < \varepsilon < 1$, finds an integral vector p and an integer q such that $||a - (1/q)p|| < \varepsilon/q$, and $1 \leq q \leq 2^{n(n+1)/4}\varepsilon^{-n}$.*

Now we pass to the description of our algorithm for $\mathrm{ILP}_{\mathbf{R}}$. As mentioned before, it consists of two main stages. In the first stage, the algorithm reduces the constraints with real coefficients to constraints with integer coefficients determining the same admissible set. The first step of this reduction is the substitution of a given real vector with an appropriate rational vector, justified by the following lemma[1].

Lemma 4. *Given a vector $\alpha \in \mathbf{R}^n$ with $|\alpha_j| \leq 1, j = 1, 2, \ldots, n$, and $D \in \mathbf{Z}_+$, there exists an $O(n^4 \log n(n + \log D))$ algorithm that finds $p \in \mathbf{Z}^n$ and $q \in \mathbf{Z}_+$ such that $|\alpha_j - p_j/q| < 1/(qD)$, $j = 1, 2, \ldots, n$, and $1 \leq q \leq \lceil 2^{n(n+5)/4} D^n \rceil$.*

Proof First we describe the algorithm finding $p \in \mathbf{Z}^n$ and $q \in \mathbf{Z}_+$ with the required properties. It consists of two basic steps.

1. For each α_j, $1 \leq j \leq n$, find the closest rational fraction a_j with denominator $G = \lceil 2^{n(n+5)/4} D^{n+1} \rceil$.
2. Apply the algorithm from Lemma 3 with input $a = (a_1, \ldots, a_n) \in \mathbf{Q}^n$ and $\varepsilon = 1/(2D)$. The output is a vector $p \in \mathbf{Z}^n$ and an integer $q \in \mathbf{Z}_+$ with $||a - (1/q)p|| < 1/(2qD)$ and $1 \leq q \leq \lceil 2^{n(n+5)/4} D^n \rceil$.

Clearly, $|\alpha_j - a_j| \leq 1/(2G)$. Then we have

$$\left|\alpha_j - \frac{p_j}{q}\right| \leq |\alpha_j - a_j| + \left|a_j - \frac{1}{q}p_j\right| \leq |\alpha_j - a_j| + \left\|a - \frac{1}{q}p\right\| <$$

$$< \frac{1}{2G} + \frac{1}{2qD} \leq \frac{1}{2.\lceil 2^{n(n+5)/4} D^n \rceil . D} + \frac{1}{2qD} \leq \frac{1}{qD},$$

i.e., the obtained vector p and integer q are as desired.

[1] To reduce the given real constraints to an equivalent set of integer constraints, one can also use the approach from [20].

Now we evaluate the algorithm's complexity. For a given real number α_j, the closest rational fraction with denominator $G = \lceil 2^{n(n+5)/4} D^{n+1} \rceil$ can be found in time $O(\log G) = O(n^2 + n \log D)$. Thus the overall time complexity of Step 1 is $O(n^3 + n^2 \log D)$.[2]

Step 2 involves the simultaneous Diophantine approximation algorithm applied to the particular class of inputs $a \in \mathbf{Q}^n$, $\varepsilon = 1/(2D)$ obtained in Step 1. As a matter of fact, this algorithm is a specialization of the Lovász basis reduction algorithm, applied to a matrix of the form

$$\begin{pmatrix} 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & 1 & a_n \\ & & & 1/G \end{pmatrix}$$

where $a_1, a_2, \ldots, a_n$ are rational numbers, all of them with the same denominator $G = 2^{O(n(n+\log D))}$. The following bound on the number of iterations holds.

Lemma 5. *(see [4, Lemma 4.4]) In Step 2 of the algorithm,* $O(\log \frac{F_{start}}{F_{end}}) = O(n^3 + n^2 \log D)$ *iterations are performed.*

From (4) and Lemma 5, we obtain that the time complexity of Step 2 is

$$O\left(n^2 \log n \log \frac{F_{start}}{F_{end}}\right) = O(n^2 \log n(n^3 + n^2 \log D)) =$$

$$= O(n^4 \log n(n + \log D)).$$

Thus the overall time complexity of the algorithm is $O(n^4 \log n(n + \log D))$. □

The algorithm of Lemma 4 can be used to substitute any *real* constraint $ax \leq b$ with an *integer* one, preserving the same admissible integer points x with $\mathbf{0} \leq x \leq d$, $d \in \mathbf{R}^n$. More precisely, we have the following lemma.

Lemma 6. *Let* $T = \{x \in \mathbf{Z}^n : ax \leq b; \mathbf{0} \leq d\}$, *where* $a \in \mathbf{R}^n$, $b \in \mathbf{R}$, $d \in \mathbf{Z}^n_+$. *Then there exists an algorithm which finds a vector* $r \in \mathbf{Z}^n$ *and a number* $r_0 \in \mathbf{Z}$ *such that* $T = \{x \in \mathbf{Z}^n : rx \leq r_0; \mathbf{0} \leq x \leq d\}$. *The algorithm involves at most n applications of the algorithm from Lemma 4, with* $D = ||d||$.

Proof of the above fact is available in [4, Lemma 5.1].

From Lemmas 4 and 6, we obtain that the overall time complexity of the reduction stage is $O(mn^5 \log n(n + \log ||d||))$. Furthermore, the bit-size of the generated integers is $O(n^2(n + \log ||d||))$. Therefore, the overall bit-size of the reduced problem is $O(mn^3(n + \log ||d||))$.

At this point, the second of the announced results follows immediately. First we unfold the applications of the Lovász basis reduction algorithm in an algebraic

[2] Note that in the BSS-model extended with a unit cost floor operation (the case handled in [4]) this requires $O(n)$ operations.

decision tree with depth $O(mn^5 \log n(n + \log ||d||))$. After that, we branch on every bit of the obtained integer data problem, which adds $O(mn^3(n + \log ||d||))$ to the depth of the tree. Thus we obtain the following theorem.

Theorem 1. *There is an* $O(mn^5 \log n(n + \log ||d||))$ *algebraic decision tree for* $ILP_{\mathbf{R}}$.

To obtain the other result of ours, we continue with the second stage of the algorithm. That stage is an application of the Lenstra's algorithm to the integer data problem obtained as output of the first stage. A recursive step of this algorithm reduces an n-dimensional problem to a set of subproblems of dimension $n - 1$, whose number is exponential but depending only on n. The basic algorithms used in this reduction are the Lovász basis reduction algorithm and the Kannan-Bachem's Hermite normal form algorithm. In addition, a linear programming problem of dimension $(m + 2n) \times n$ is to be solved.

The two algorithms are applied to matrices of dimension depending only on n and with entries of bit-size $O(\log ||d||)$, as the value of n is fixed. Then, by Lemmas 1 and 2, their complexity as well as the bit-size of the integers they generate, are bounded by $O(\log ||d||)$. The linear programming problem can be solved in time $O(m+n)$ (i.e., *linear* in m) using the well-known Megiddo's algorithm [16]. Hence, if n is fixed, the overall complexity of this stage is $O(m \log ||d||)$. Thus we have obtained the following theorem.

Theorem 2. *There is an* $O(m \log ||d||)$ *algorithm for* $ILP_{\mathbf{R}}$ *of fixed dimension* n.

Theorem 2 implies a tight bound for the algebraic complexity of the knapsack problem.

Corollary 1. *The algebraic complexity of the knapsack problem* $KP_{\mathbf{R}}$ *of fixed dimension* n *is* $\Theta(\log \frac{1}{a_{\min}})$.

Proof An upper bound $O(\log \frac{1}{a_{\min}})$ follows from Theorem 2. A lower bound $\Omega(\log \frac{1}{a_{\min}})$ follows from [5], where a tight bound $\Theta(\log \frac{1}{a_{\min}})$ is proved for the algebraic complexity of the two-dimensional knapsack problem with real coefficients. □

Regarding possible practical applications, one can expect that the proposed algorithm for $\mathrm{ILP}_{\mathbf{R}}$ may be useful for problems with a small number of variables and a large number of constraints.

4 Lower Bound for the Knapsack Problem

In this section we study the complexity of the Boolean knapsack problem

$(0/1 - \mathrm{KP}_{\mathbf{R}})$ Given $a \in \mathbf{R}^n_+$, decide if there is $x \in \{0,1\}^n$ such that $a^T x = 1$.

In the classical setting, the knapsack problem has been studied intensively (see [15] and the bibliography therein). In particular, the problem is NP-complete

[8]. Regarding "real" knapsacks, a number of results have been proved. Notable among them are the lower bounds $\Omega(n^2 \log \frac{1}{a_{\min}})$ and $\Omega(n^2)$ for $\mathrm{KP}_{\mathbf{R}}$'s and 0/1-$\mathrm{KP}_{\mathbf{R}}$'s complexity, respectively (see [1]). In [3] the topological complexity of the latter problem is found. [17] provides a parallel time lower bound. Some other results are presented in [7,4,5]. For a related discussion the reader is also referred to [2].

In this section, we take one more step towards determining the algebraic complexity of the knapsack problem. We obtain a result which complements the above mentioned Ben-Or's lower bounds. More precisely, we have the following theorem.

Theorem 3. *No algorithm solving* $0/1{-}\mathrm{KP}_{\mathbf{R}}$ *can achieve a time complexity* $o(n \log n) \cdot f(a_1, \ldots, a_n)$*, where* f *is an arbitrary continuous function of* n *variables.*

Remark 1. The requirement for f to be a continuous function is essential, as follows from [6]. More precisely, it has been shown that there is an $O(n \frac{1}{\delta(a)})$ algorithm for $0/1{-}\mathrm{KP}_{\mathbf{R}}$, where $\delta(a) = \min\{|a^T z| : a^T z \neq 0, z \in \{-1, 0, 1\}^n\}$.

Proof Assume the opposite, i.e., that there is an algorithm $\mathcal{A}$ that solves 0/1-$\mathrm{KP}_{\mathbf{R}}$ in $o(n \log n) \cdot f(a_1, \ldots, a_n)$ time, for a continuous function f. We consider a subclass C of inputs of 0/1-$\mathrm{KP}_{\mathbf{R}}$ determined by the constraints

$$\frac{1}{3} < a_i < \frac{2}{3} \quad \text{for } 1 \leq i \leq n.$$

Let us denote

$$\begin{aligned} C &= \left\{a \mid \tfrac{1}{3} < a_i < \tfrac{2}{3},\ 1 \leq i \leq n\right\}, \\ C_{yes} &= \left\{a \mid a \in C,\ \exists x \in \{0,1\}^n : a^T x = 1\right\}, \\ C_{no} &= C \setminus C_{yes}. \end{aligned}$$

For any problem input from the considered subclass the value $f(a_1, \ldots, a_n)$ is bounded by a constant, and thus the time complexity of the algorithm $\mathcal{A}$ reduces to $o(n \log n)$.

On the other hand, according to the Ben-Or's theorem [1], in the considered computational model a lower bound $\Omega(\log \#c.c.(C_{no}))$ holds for the complexity of any algorithm solving 0/1-$\mathrm{KP}_{\mathbf{R}}$ for inputs from C_{yes}, where $\#c.c.(C_{no})$ is the number of connected components of C_{no}. We will show that $\#c.c.(C_{no}) = n!$, i.e., $\log \#c.c.(C_{no}) = \log n! = \Theta(n \log n)$. This will contradict the $o(n \log n)$ time complexity of the algorithm $\mathcal{A}$ on inputs from C. Thus, to complete the proof it suffices to prove the following lemma.

Lemma 7. *The set* C_{no} *has exactly* $n!$ *connected components.*

Proof First of all, let us observe that $a \in C_{yes}$ if and only if $\exists i, j\ i \neq j$ such that $a_i + a_j = 1$. For every $a \in C_{no}$ we denote

$$S_a = \{\{a_i, a_j\} \mid i \neq j,\ a_i + a_j < 1\}.$$

As a first step of the proof we will show that C_{no} has as many connected components as the number of all distinct sets S_a, $a \in C_{no}$.

First we show that if for some $a', a'' \in C_{no}$ the condition $S_{a'} = S_{a''}$ holds, then a' and a'' belong to the same connected component of C_{no}. Clearly, $C_{no} = \left\{a \mid a \in C,\ \forall x \in \{0,1\}^n : a^T x \neq 1\right\}$. Since $S_{a'} = S_{a''}$, for every i, j $i \neq j$ both $a'_i + a'_j - 1$ and $a''_i + a''_j - 1$ have the same sign, either positive or negative but not zero. The same is the sign of $a_i + a_j - 1$, where $a = (a_1, \ldots, a_n)$ is any affine combination of a' and a''. Thus $a \in C_{no}$, so that the whole segment with endpoints a' and a'' lies in C_{no}. This implies that these two points belong to the same connected component.

Now we prove that if $a', a'' \in C_{no}$ are in the same connected component, then $S_{a'} = S_{a''}$. Assume the opposite, i.e., that there are $a', a'' \in C_{no}$ belonging to the same connected component D, while $S_{a'} \neq S_{a''}$. Then there are i, j $i \neq j$ such that $a'_i + a'_j < 1$ and $a''_i + a''_j > 1$. Let $\mathcal{L}$ be a continuous curve with end-points a' and a'', which is contained in D. We define a function $h(x) = x_i + x_j$, where x_i, x_j are, respectively, the ith and jth component of $x \in \mathbf{R}^n$. Let us consider the restriction of $h(x)$ on the curve $\mathcal{L}$. We have $h(a') = a'_i + a'_j < 1 < a''_i + a''_j = h(a'')$. Since h is continuous, there must be a point a on the curve $\mathcal{L}$ for which $h(a) = a_i + a_j = 1$. But $a \in \mathcal{L} \subseteq D \subseteq C_{no}$ and therefore $a_i + a_j < 1$, which is a contradiction.

Thus it only remains to count all distinct sets S_a, $a \in C_{no}$. We will use induction on the dimension n. As the basis of the induction, for $n = 2$ we have two such distinct sets, namely $\emptyset$ and $\{\{a_1, a_2\}\}$. Suppose that the thesis is true for dimension $n - 1$, and take an arbitrary set S_a, where $a = (a_1, \ldots, a_{n-1})$ is an $(n-1)$-dimensional vector. Without loss of generality we assume that the coordinates of a are all distinct, and consider them in an increasing order:

$$\frac{1}{3} < a_{i_1} < a_{i_2} < \ldots < a_{i_{n-1}} < \frac{2}{3}.$$

To pass to dimension n, we need to add one more coordinate a_n to the vector a. One can choose a_n in n different ways:

$$\begin{array}{l} 1 - a_{i_1} < a_n < \frac{2}{3}, \\ 1 - a_{i_j} < a_n < 1 - a_{i_{j-1}} \text{ for } 2 \leq j \leq n-1, \\ \frac{1}{3} < a_n < 1 - a_{i_{n-1}}. \end{array}$$

Thus we get n distinct sets "generated" by S_a:

$$S_a \cup \left\{\{a_{i_k}, a_n\} \mid 1 \leq k \leq j\right\}, \quad \text{for } 0 \leq j \leq n-1.$$

Note also that two sets generated by sets $S_{a'} \neq S_{a''}$ are distinct because their maximum subsets of (unordered) pairs not containing a_n are (i.e., the sets $S_{a'}$ and $S_{a''}$, themselves).

Thus we have proved that the number of connected components of C_{no} increases by a factor of n when passing from dimension $n - 1$ to dimension n. Hence, this number is exactly $n!$, as claimed. □

Remark 2. The Ben-Or's theorem states a lower bound $\Omega(\log \#c.c.(C_{no}))$ on the complexity of any algorithm solving 0/1-KP$_{\mathbf{R}}$. Note that, in the general case, when no restrictions on the coefficients $a_1, a_2, \ldots, a_n$ are imposed, the number of connected components of C_{no} is $2^{\Omega(n^2)}$. Hence, the lower bound $\Omega(n^2)$ holds. This larger number $2^{\Omega(n^2)}$ (compared to the number $n!$ of connected components of the class C) is due to the inputs in which the a_i's are "close to 0 or 1." These inputs are excluded from C. Therefore, the Ben-Or's bound $\Omega(n^2)$ does not imply directly ours, although we have used his theorem, together with Lemma 7, to obtain ours.

Remark 3. To obtain the complexity result of Theorem 3, we have used the class of instances $C = \left\{a \mid \frac{1}{3} < a_i < \frac{2}{3},\ 1 \leq i \leq n\right\}$. It is easy to see that an $O(n \log n)$ algorithm exists for this particular class. To show this, first we sort in $O(n \log n)$ time the coefficients of $a^T x = 1$. After an appropriate substitution and enumeration of the variables we obtain an equation with coefficients $a_1 < a_2 < \ldots < a_n$. As already observed, a solution to $a^T x = 1$ exists if and only if $\exists i, j\ i \neq j$ such that $a_i + a_j = 1$. In order to check whether this condition is met, we search the sorted array of coefficients in linear time, as follows. Set $i = 1$, $j = n$. If $a_i + a_j < 1$, then set $i := i + 1$; If $a_i + a_j > 1$, then set $j := j - 1$. If $a_i + a_j = 1$ or $i = j$, then stop. In the former case the equation has a solution (namely, $x_i = 1$, $x_j = 1$, $x_k = 0$ for $k \neq i, j$). In the latter case a solution does not exist. The complexity of this procedure is $O(n)$, and thus the overall complexity of the algorithm is $O(n \log n)$. The proof of Theorem 3 demonstrates that for the class C this algorithm is, in fact, optimal.

Clearly, an analogous result holds for the complexity of the classical Boolean knapsack problem (0/1-KP) $a^T x = b$ with integer coefficients. This problem is equivalent to the equation $\bar{a}^T x = 1$ with $\bar{a} = \frac{1}{b} a$, to which Theorem 3 applies. Thus, we have lower time complexity bounds, both for the Boolean knapsack problem with real coefficients and the classical formulation, and these bounds are independent of the known lower bound $\Omega(n^2)$.

One can show that the result of Theorem 3 is still valid for the integer knapsack problem KP$_{\mathbf{R}}$. Unlike the Boolean case, here an input a belongs to C_{yes} not only if $a_i + a_j = 1$ for some indices i, j, but also if $a_k = \frac{1}{2}$ for some index k. Accordingly, the set S_a is modified as

$$S_a = \{\{a_i, a_j\} \mid\ a_i + a_j < 1\}.$$

Note in the definition above that we allow $i = j$. This adds to the set S_a every singleton $\{a_k\}$ such that $a_k = 1/2$. One can show that C_{no} has *at least* as many connected components as the number of the distinct sets S_a, $a \in C_{no}$. Then by induction on n one obtains that the set C_{no} has *at least* $n!$ connected components, from where the result follows. The proof is a straightforward modification of the one of Theorem 3 and therefore is omitted.

As a last comment of this section we mention that the Ben-Or's lower bound, as well as the negative complexity result of Theorem 3, suggest seeking certain

approximation solutions to the knapsack problem. For instance, one can look for a Boolean vector that is "enough close" to the hyperplane determined by $a^T x = 1$ (e.g., minimizing $|a^T x - 1|$ within a given tolerance $\varepsilon > 0$). We notice that the best of the existing algorithms (see, e.g., [11]) for this approximation problem are indeed *linear* with respect to the dimension n.

5 Concluding Remarks

We have presented an $O(m \log ||d||)$ algorithm for integer linear programming with real coefficients and fixed number of variables, within the Blum-Shub-Smale computational model. A further task would be to show that this complexity bound is tight.

We have also obtained a lower bound for the complexity of the real knapsack problem. In view of this result, it would be interesting to know if there is an algorithm for 0/1-$\mathrm{KP}_{\mathbf{R}}$ with time complexity $O(n^{2-\delta} f(a))$, $\delta \geq 0$.

Some of the obtained results (e.g., Corollary 1) show that the integer programming problems are, in general, intractable in the framework of the complexity theory over the reals, since their complexity cannot be bounded by any polynomial in the input size, the latter being a polynomial only in m and n. Some further refinements of the theory suggest, however, that these problems can be considered as efficiently solvable. Following Smale [23], a numerical algorithm can be considered as efficient only if its complexity is bounded by a polynomial in the problem dimensions and the logarithm of its *weight*. The weight function is defined in accordance with the problem specificity and used to measure the difficulty of a problem instance. Under such a convention, let us define the weight of $\mathrm{ILP}_{\mathbf{R}}$ as a number $||d||$ which bounds the norms of the admissible solutions. Then the results of Section 3 imply that $\mathrm{ILP}_{\mathbf{R}}$ and $\mathrm{KP}_{\mathbf{R}}$ are efficiently solvable in the above sense.

Acknowledgments

The authors thank Bruno Codenotti for reading an early version of this paper, and the two referees for their valuable suggestions.

References

1. Ben-Or, M.: Lower Bounds for Algebraic Computation Tree. Proc. Of the 15th ACM STOC (1983) 80–86
2. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer-Verlag, Berlin Heidelberg New York (1995)
3. Blum, L., Shub, M., Smale, S.: On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines. Bull. Amer. Math. Soc. (NS), **21** (1989) 1–46
4. Brimkov, V.E., Danchev, S.S.: Real Data – Integer Solution Problems within the Blum-Shub-Smale Computational Model: J. of Complexity, **13** (1997) 279–300

5. Brimkov, V.E., Danchev, S.S., Leoncini, M.: Tight Complexity Bounds for the Two-Dimensional Real Knapsack Problem. Calcolo, **36** (1999) 123–128
6. Brimkov, V.E., Dantchev, S.S.: Lower Bounds, "Pseudopolynomial" and Approximation Algorithms for the Knapsack Problem with Real Coefficients. Electronic Colloquium on Computational Complexity, TR98-015 (1998). http://www.eccc.uni-trier.de/eccc/
7. Cucker, F., Shub, M.: Generalized Knapsack Problem and Fixed Degree Separations. Theoret. Comput. Sci., **161** (1996) 301–306
8. Garey, M.S., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. Freeman & Co., San Francisco (1979)
9. Hastad, J., Just, B., Lagarias, J.C., Schnoor, C.P.: Polynomial Time Algorithms for Finding Integer Relations among Real Numbers. SIAM J. Comput., **18** (1989) 859–881
10. Kannan, R., Bachem, A.: Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix. SIAM J. Comput., **8** (1979) 499–507
11. Kellerer, H., Mansini, R., Pferschy, U., Speranza, M.G.: An Efficient Fully Polynomial Approximation Scheme for the Subset-Sum Problem. Proc. 8th ISAAC Symposium, Lecture Notes in Computer Science, Vol. 1350. Springer-Verlag, Berlin Heidelberg New York (1997) 394–403
12. Lenstra, H.W., Jr.: Integer Programming with a Fixed Number of Variables. Math. Oper. Res., **8** (1983) 538–548
13. Lenstra, A.K., Lenstra, H.W., Jr., Lovász, L.: Factoring Polynomials with Rational Coefficients. Math. Ann., **261** (1982) 515–534
14. Meyer Auf Der Heide, F.: A Polynomial Linear Search Algorithm for the n-Dimensional Knapsack Problem. J. of ACM, **31** (3) (1984) 668–676
15. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations. Wiley, Chichester New York Brisbane Toronto Singapore (1990)
16. Megiddo, N.: Linear Programming in Linear Time when the Dimension is Fixed. J. of ACM, **31** (1) (1984) 114–127
17. Montaña, J.L., Pardo, L.M.: Lower Bounds for Arithmetic Networks. Appl. Algebra Eng. Comm. Comput., **4** (1993) 1–24
18. Novak, E., The Real Number Model in Numerical Analysis. J. of Complexity, **11** (1994) 57–73
19. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer-Verlag, Berlin Heidelberg New York (1985)
20. Rössner, C., Schnorr, C.P.: An Optimal, Stable Continued Fraction Algorithm for Arbitrary Dimension. In: Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science, Vol. 1084. Springer-Verlag, Berlin Heidelberg New York (1996) 31–43
21. Schrijver, A.: Theory of Linear and Integer Programming, Wiley, Chichester New York Brisbane Toronto Singapore (1986)
22. Strassen, V.: Algebraic Complexity Theory. In: van Leeuwen, J. (Ed.): Handbook of Theoretical Computer Science, Vol. A, Elsevier, Amsterdam (1990) 633–672
23. Smale, S.: Some Remarks on the Foundations of Numerical Analysis. SIAM Rev., **32** (2) (1990) 211–220

On Logarithmic Simulated Annealing*

Andreas Albrecht[1,2] and Chak-Kuen Wong[2]**

[1] University of Hertfordshire
Dept. of Computer Science, Hatfield, Herts AL10 9AB, UK
[2] Dept. of Computer Science and Engineering,
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong

Abstract. We perform a convergence analysis of simulated annealing for the special case of logarithmic cooling schedules. For this class of simulated annealing algorithms, B. Hajek [7] proved that the convergence to optimum solutions requires the lower bound $\Gamma/\ln(k+2)$ on the cooling schedule, where k is the number of transitions and Γ denotes the maximum value of the escape depth from local minima. Let n be a uniform upper bound for the number of neighbours in the underlying configuration space. Under some natural assumptions, we prove the following convergence rate: After $k \geq n^{O(\Gamma)} + \log^{O(1)}(1/\varepsilon)$ transitions the probability to be in an optimum solution is larger than $(1-\varepsilon)$. The result can be applied, for example, to the average case analysis of stochastic algorithms when estimations of the corresponding values Γ are known.

1 Introduction

Simulated annealing was introduced independently by Kirkpatrick et al. [9] and V. Černy [6] as a new class of algorithms computing approximate solutions of combinatorial optimisation problems. The general approach itself is derived from Metropolis' method [10] to calculate equilibrium states for substances consisting of interacting molecules.

Simulated annealing algorithms can be distinguished by the method that determines how the temperature is lowered at different times of the computation process (the so-called cooling schedule). When the temperature is kept constant for a (large) number of steps, one can associate a homogeneous Markov chain with this type of computation. Under some natural assumptions, the probabilities of configurations tend to the Boltzmann distribution in the case of homogeneous Markov chains (see [13,14,15]). This class of simulated annealing

* Research partially supported by the Strategic Research Programme at The Chinese University of Hong Kong under Grant No. SRP 9505 and by a Hong Kong Government RGC Earmarked Grant, Ref. No. CUHK 4010/98E.

** On leave from IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y., U.S.A.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 301–314, 2000.
© Springer-Verlag Berlin Heidelberg 2000

algorithms has been studied intensely and numerous heuristics have been derived for a wide variety of combinatorial optimisation problems (see [2,5,11,12,16]).

When the temperature is lowered at any step, the probabilities of configurations are computed from an inhomogeneous Markov chain. The special case of logarithmic cooling schedules has been investigated in [1,4,5,7]. B. Hajek [7] proved that logarithmic simulated annealing tends to an optimum solution if and only if the cooling schedule is lower bounded by $\Gamma/\ln(k+2)$, where Γ is the maximum value of the escape depth from local minima of the underlying energy landscape. O. Catoni [4,5] obtained a convergence rate of $M \cdot k^{-\alpha}$, where M depends on the number of configurations with a certain value of the objective function and α is related to the parameter Γ. Since structural parameters are treated as constants, the estimation does not provide a time bound that depends on Γ and the confidence parameter ε only.

Given the configuration space $\mathcal{F}$, let $\mathbf{a}_f(k)$ denote the probability to be in configuration f after k steps of an inhomogeneous Markov chain. The problem is to find a lower bound for k such that $\sum_{f\in\mathcal{F}_{\min}} \mathbf{a}_f(k) > 1-\varepsilon$ for $f \in \mathcal{F}_{\min} \subseteq \mathcal{F}$ minimising the objective function. By n we denote a uniform upper bound for the number of neighbours of configurations $f \in \mathcal{F}$.

We obtain a run-time of $n^{O(\Gamma)} + \log^{O(1)} 1/\varepsilon$ that is sufficient to approach with probability $1-\varepsilon$ the minimum value of the objective function. The present approach is based on a very detailed analysis of transition probabilities and has been developed in the context of equilibrium computations for specific physical systems [3].

2 Preliminaries

Simulated annealing algorithms are acting within a configuration space in accordance with a certain neighbourhood relation, where the particular transitions between adjacent elements of the configuration space are governed by an objective function. In the following section we introduce these notations for our problem setting.

The configuration space is *finite* and denoted by $\mathcal{F}$. We assume an objective function $\mathcal{Z} : \mathcal{F} \longrightarrow \mathbb{N}$ that for simplicity takes it values from the set of integers. By $\mathcal{N}_f$ we denote the set of neighbours of f, including f itself.

We assume that $\mathcal{F}$ is *reversible*: Any transition $f \to f'$, $f' \in \mathcal{N}_f$, can be performed in the reverse direction, i.e., $f \in \mathcal{N}_{f'}$. We set $n := \max_{f\in\mathcal{F}} |\mathcal{N}_f|$.

The set of minimal elements (optimum solutions) is defined by

$$\mathcal{F}_{\min} := \big\{\, f \,:\, \forall f'\big(f' \in \mathcal{F} \to\ \mathcal{Z}(f) \,\le\, \mathcal{Z}(f')\big)\big\}.$$

Example We consider Boolean conjunctive terms defined on n variables. The conjunctions are of length $\lceil \log n \rceil$ and have to be guessed from negative examples and one positive example $\tilde{\sigma}$. Hence, $\mathcal{F}$ consists of $t_{\tilde{\sigma}} = x^{\tilde{\sigma}}$ and all subterms t that can be obtained by deleting a literal from $x^{\tilde{\sigma}}$ and that reject all negative examples. Each neighbourhood $\mathcal{N}_t$ contains at most $n' \le n+1$ elements, since deleted literals can be included again. The configuration space is therefore reversible. The objective function

is given by the length of terms t and in this case the optimum in known and equal to $\lceil \log n \rceil$.

In simulated annealing, the transitions between neighbouring elements are depending on the objective function $\mathcal{Z}$. Given a pair of configurations $[f, f']$, $f' \in \mathcal{N}_f$, we denote by $G[f,\, f']$ the probability of generating f' from f and by $A[f, f']$ the probability of accepting f' once it has been generated from f. Since we consider a single step of transitions, the value of $G[f,\, f']$ depends on the set $\mathcal{N}_f$. As in most applications of simulated annealing, we take a uniform probability which is given by

$$G[f,\, f'] := \frac{1}{|\mathcal{N}_f\,|}. \tag{1}$$

The acceptance probabilities $A[f,\, f']$, $f' \in \subseteq \mathcal{F}$ are derived from the underlying analogy to thermodynamic systems [2,10]:

$$A[f, f'] := \begin{cases} 1, & \text{if } \mathcal{Z}(f') - \mathcal{Z}(f) \leq 0, \\ e^{-\frac{\mathcal{Z}(f') - \mathcal{Z}(f)}{c}}, & \text{otherwise,} \end{cases} \tag{2}$$

where c is a control parameter having the interpretation of a *temperature* in annealing procedures.

Finally, the probability of performing the transition between f and f' is defined by

$$\mathbf{Pr}\{f \to f'\} = \begin{cases} G[f,\, f'] \cdot A[f,\, f'], & \text{if } f' \neq f, \\ 1 - \sum\limits_{f' \neq f} G[f,\, f'] \cdot A[f,\, g], & \text{otherwise.} \end{cases} \tag{3}$$

By definition, the probability $\mathbf{Pr}\{f \to f'\}$ depends on the control parameter c.

Let $\mathbf{a}_f(k)$ denote the probability of being in the configuration f after k steps performed for the same value of c. The probability $\mathbf{a}_f(k)$ can be calculated in accordance with

$$\mathbf{a}_f(k) := \sum_h \mathbf{a}_h(k-1) \cdot \mathbf{Pr}\{h \to f\}. \tag{4}$$

The recursive application of (4) defines a Markov chain of probabilities $\mathbf{a}_f(k)$, where $f \in \mathcal{F}$ and $k = 1, 2, \ldots$. If the parameter $c = c(k)$ is a constant c, the chain is said to be a *homogeneous* Markov chain; otherwise, if $c(k)$ is lowered at any step, the sequence of probability vectors $\mathbf{a}(k)$ is an *inhomogeneous* Markov chain.

In the present paper we are focusing on a special type of inhomogeneous Markov chains where the value $c(k)$ changes in accordance with

$$c(k) = \frac{\Gamma}{\ln(k+2)},\ k = 0, 1, \ldots. \tag{5}$$

The choice of $c(k)$ is motivated by Hajek's Theorem [7] on logarithmic cooling schedules for inhomogeneous Markov chains. To explain Hajek's result, we first

need to introduce some parameters characterising local minima of the objective function:

Definition 1 *A configuration $f' \in \mathcal{F}$ is said to be reachable at height h from $f \in \mathcal{F}$, if $\exists f_0, f_1, \ldots, f_r \in \mathcal{F}\big(f_0 = f \wedge f_r = f'\big)$ such that $G[f_u, f_{u+1}] > 0$, $u = 0, 1, \ldots, (r-1)$ and $\mathcal{Z}(f_u) \leq h$, for all $u = 0, 1, \ldots, r$.*

We use the notation *height*$(f \Rightarrow f') \leq h$ for this property. The function f is a *local minimum*, if $f \in \mathcal{F} \setminus \mathcal{F}_{\min}$ and $\mathcal{Z}(f') > \mathcal{Z}(f)$ for all $f' \in \mathcal{N}_f \setminus f$.

Definition 2 *Let $g_{\min}$ denote a local minimum, then depth$(g_{\min})$ denotes the smallest h such that there exists a $g' \in \mathcal{F}$, where $\mathcal{Z}(g') < \mathcal{Z}(g_{\min})$, which is reachable at height $\mathcal{Z}(g_{\min}) + h$.*

The following convergence property has been proved by B. Hajek:

Theorem 1 [7] *Given a cooling schedule defined by*

$$c(k) = \frac{\Gamma}{\ln(k+2)},\ k = 0, 1, \ldots,$$

the asymptotic convergence $\sum_{f \in \mathcal{F}} \mathbf{a}_f(k) \underset{k \to \infty}{\longrightarrow} 1$ of the stochastic algorithm that is based on (2) and (3) is guaranteed if and only if

(i) $\forall g, g' \in \mathcal{F}\, \exists g_0, g_1, \ldots, g_r \in \mathcal{F}\big(g_0 = g \wedge g_r = g'\big)$:
$G[g_u, g_{u+1}] > 0$, $u = 0, 1, \ldots, (r-1)$;
(ii) $\forall h$: *height*$(f \Rightarrow f') \leq h \iff$ *height*$(f' \Rightarrow f) \leq h$;
(iii) $\Gamma \geq \max_{g_{\min}}$ *depth*$(g_{\min})$.

In the following, we assume that the conditions (i), ... , (iii) of Hajek's Theorem are satisfied for our configurations space $\mathcal{F}$. Furthermore, we set the upper bound $|\mathcal{F}| \leq \exp(n^{O(1)})$ which is common in combinatorial optimisation. Additionally, we assume that the difference of the objective function is the same between neighbouring elements, like in the case of our Example.

3 Convergence Analysis

Our convergence result is derived from a careful analysis of the "exchange of probabilities" among configurations that belong to adjacent distance levels, where the distance is measured by the minimum number of steps to reach an element of $\mathcal{F}_{\min}$. Therefore, in addition to the value of the objective function, the elements of the configuration space are further distinguished by their distance to $\mathcal{F}_{\min}$.

We introduce the following partition of the set of configurations with respect to the value of the objective function:

$$L_0 := \mathcal{F}_{\min} \quad \text{and } L_{h+1} := \{f : f \in \mathcal{F} \wedge \forall f' \big(f' \in \mathcal{F} \setminus \bigcup_{i=0}^{h} L_i \rightarrow \mathcal{Z}(f') \geq \mathcal{Z}(f)\}.$$

The highest level within $\mathcal{F}$ is denoted by L_{h_m} and we set

$$\begin{aligned} s(f) &:= |\{ f' : f' \in \mathcal{N}_f \wedge \mathcal{Z}(f') > \mathcal{Z}(f)\}|, \\ q(f) &:= |\{ f' : f' \in \mathcal{N}_f \wedge \mathcal{Z}(f') = \mathcal{Z}(f)\}|, \\ r(f) &:= |\{ f' : f' \in \mathcal{N}_f \wedge \big(\mathcal{Z}(f') < \mathcal{Z}(f)\big) \}| \, . \end{aligned}$$

As will be seen later, the $q(f)$ neighbors with the same value of the objective function can be treated in the same way as f itself (we have $q(f) = 0$ in our Example of Boolean conjunctions). Therefore, we will not distinguish between f and these neighbors, see Lemma 2 and the following equations. The introduced notations imply

$$s(f) + q(f) + r(f) = |\mathcal{N}_f| - 1. \tag{6}$$

We consider the probability $\mathbf{a}_f(k)$ to be in the configuration $f \in \mathcal{F}$ after k steps of an *inhomogeneous* Markov chain, i.e., the "temperature" is defined in accordance with (5). We observe that

$$\frac{1}{\big(k+2\big)^{(\mathcal{Z}(f)-\mathcal{Z}(f'))/\Gamma}} = e^{-\frac{\mathcal{Z}(f)-\mathcal{Z}(f')}{c(k)}}, \; k \geq 0. \tag{7}$$

To simplify notations we take a new objective function with the same notation: $\mathcal{Z}(f) := \mathcal{Z}(f)/\Gamma$.

By using (1) till (3), one obtains from (4) by straightforward calculations

$$\begin{aligned} \mathbf{a}_f(k) &= \sum_{f' \in \mathcal{N}_f} \mathbf{a}_{f'}(k-1) \cdot \mathbf{Pr}_{c(k)}\{f' \rightarrow f\} \\ &= \mathbf{a}_f(k-1) \cdot \mathbf{Pr}_{c(k)}\{f \rightarrow f\} + \sum_{f' \neq f} \mathbf{a}_{f'}(k-1) \cdot \mathbf{Pr}_{c(k)}\{f' \rightarrow f\} \\ &= \mathbf{a}_f(k-1) \cdot \Big(1 - \sum_{f' \neq f} \mathbf{Pr}_{c(k)}\{f \rightarrow f'\}\Big) + \\ &\quad + \sum_{f' \neq f} \mathbf{a}_{f'}(k-1) \cdot \mathbf{Pr}_{c(k)}\{f' \rightarrow f\}. \end{aligned}$$

We employ (7), and the resulting equation is given by

$$\mathbf{a}_f(k) = \left(\frac{s(f)+1}{|\mathcal{N}_f|} - \sum_{\substack{i=1 \\ \mathcal{Z}(f_i) \geq \mathcal{Z}(f)}}^{s(f)+q(f)} \frac{\big(k+1\big)^{-(\mathcal{Z}(f_i)-\mathcal{Z}(f))}}{|\mathcal{N}_f|} \right) \cdot \mathbf{a}_f(k-1) + \tag{8}$$

$$+ \sum_{\substack{i=1 \\ \mathcal{Z}(f_i) \geq \mathcal{Z}(f)}}^{s(f)+q(f)} \frac{\mathbf{a}_{f_i}(k-1)}{\mid \mathcal{N}_{f_i} \mid} +$$

$$+ \sum_{\substack{j=1 \\ \mathcal{Z}(f'_j) < \mathcal{Z}(f)}}^{r(f)} \frac{\mathbf{a}_{f_j}(k-1)}{\mid \mathcal{N}_{f_j} \mid} \cdot \frac{1}{\big(k+1\big)^{\mathcal{Z}(f)-\mathcal{Z}(f_j)}}.$$

This representation (expansion) will be used in the following as the main relation reducing $\mathbf{a}_f(k)$ to probabilities from previous steps.

Given $f \in \mathcal{F}$, we consider a shortest path of length $\mathrm{dist}(f)$ from f to $\mathcal{F}_{\min}$, i.e., $f' \in \mathcal{F}_{\min}$. We introduce a partition of $\mathcal{F}$ with respect to $\mathrm{dist}(f)$:

$$f \in M_i \Longleftrightarrow \mathrm{dist}(f) = i \geq 1, \quad \text{and} \quad \mathcal{M}_{\mathbf{s}} = \bigcup_{i=0}^{\mathbf{s}} M_i,$$

where $M_0 := L_0 = \mathcal{F}_{\min}$. Thus, we distinguish between distance levels M_i related to the minimal number of transitions required to reach an element of $\mathcal{F}_{\min}$ and the levels L_h that are defined by the objective function.

We suppose that $k \geq \mathbf{s}$, and we are going backwards from the k^{th} step. Our aim is to reduce the value

$$\sum_{f \notin M_0} \mathbf{a}_f(k)$$

to the sum of probabilities from previous steps. First, we consider the expansion of only a particular probability $\mathbf{a}_f(k)$ as shown in (8). At the same step k, the neighbors of f are generating terms containing $\mathbf{a}_f(k-1)$ as a factor, in the same way, as $\mathbf{a}_f(k)$ generates terms with factors $\mathbf{a}_{f_i}(k-1)$ and $\mathbf{a}_{f_j}(k-1)$. Since we consider the entire sum $\sum_f \mathbf{a}_f(k)$, the terms corresponding to a particular $\mathbf{a}_f(k-1)$ can be collected together in the expansion (8) to form a single term. Therefore, we obtain

$$(9) \quad \mathbf{a}_f(k-1) \cdot \Bigg\{ \Bigg(\frac{\mid \mathcal{N}_f \mid -r(f)}{\mid \mathcal{N}_f \mid} - \sum_{\substack{i=1 \\ \mathcal{Z}(f_i) \geq \mathcal{Z}(f)}}^{s(f)+q(f)} \frac{1}{\mid \mathcal{N}_f \mid} \cdot \frac{1}{\big(k+1\big)^{\mathcal{Z}(f_i)-\mathcal{Z}(f)}} \Bigg) +$$

$$+ \sum_{\substack{i=1 \\ \mathcal{Z}(f_i) > \mathcal{Z}(f)}}^{s(f)} \frac{1}{\mid \mathcal{N}_f \mid} \cdot \frac{1}{\big(k+1\big)^{\mathcal{Z}(f_i)-\mathcal{Z}(f)}} + \sum_{\substack{j=1 \\ \mathcal{Z}(f_j) < \mathcal{Z}(f)}}^{r(f)} \frac{1}{\mid \mathcal{N}_f \mid} \Bigg\}$$

$$= \mathbf{a}_f(k-1).$$

However, it is important to distinguish between elements from M_1 and elements from M_i, $i \geq 2$: If $f \in M_1$, the neighbors from M_0 do not generate terms containing probabilities from higher levels because the $\mathbf{a}_{f'}(k-i)$ are *not* expanded during the particular steps of the recursive procedure since they are not present

in the sum $\sum_{f \notin M_0} \mathbf{a}_f(k)$. If $f \in M_1$, the terms related to $\mathcal{Z}(f_j) < \mathcal{Z}(f)$, where $f_j \in M_0$, are missing, and therefore the following arithmetic term is generated:

$$\Big(1 - \frac{r'(f)}{\mid \mathcal{N}_f \mid}\Big) \cdot \mathbf{a}_f(k-1), \tag{10}$$

where $r'(f) \leq r(f)$ is the number of neighbors from M_0. We introduce the following abbreviations:

$$\varphi(f', f, v) := \frac{1}{\big(k+2-v\big)^{\mathcal{Z}(f')-\mathcal{Z}(f)}}, \quad D_f(k-v) := \frac{s(f)+1}{\mid \mathcal{N}_f \mid} - \sum_{i=1}^{s(f)} \frac{\varphi(f', f, v)}{\mid \mathcal{N}_f \mid}.$$

The relations expressed in (8) till (10) can be summarized to

Lemma 1 *A single step of the expansion of* $\sum_{f \notin M_0} \mathbf{a}_f(k)$ *results in*

$$\sum_{f \notin M_0} \mathbf{a}_f(k) = \sum_{f \notin M_0} \mathbf{a}_f(k-1) - \sum_{f \in M_1} \frac{r'(f)}{\mid \mathcal{N}_f \mid} \cdot \mathbf{a}_f(k-1) + \\ + \sum_{f \in M_1} \sum_{\substack{j=1 \\ f_j \in M_0 \cap \mathcal{N}_f}}^{r'(f)} \frac{\varphi(f, f_j, 1)}{\mid \mathcal{N}_{f_j} \mid} \cdot \mathbf{a}_{f_j}(k-1).$$

Sketch of Proof: We obtain from (9) and (10):

$$\begin{aligned} \sum_{f \notin M_0} \mathbf{a}_f(k) &= \sum_{f \notin M_0 \cup M_1} \mathbf{a}_f(k) + \sum_{f \in M_1} \mathbf{a}_f(k) \\ &= \sum_{f \notin M_0 \cup M_1} \mathbf{a}_f(k-1) + \sum_{f \in M_1} \Big(1 - \frac{r'(f)}{\mid \mathcal{N}_f \mid}\Big) \cdot \mathbf{a}_f(k-1) + \\ &\quad + \sum_{f \in M_1} \sum_{\substack{j=1 \\ f_j \in M_0 \cap \mathcal{N}_f}}^{r'(f)} \frac{\varphi(f, f_j, 1)}{\mid \mathcal{N}_{f_j} \mid} \cdot \mathbf{a}_{f_j}(k-1). \end{aligned}$$

The sum containing $\varphi(f, f_j, 1)/ \mid \mathcal{N}_{f_j} \mid$ is generated from $\mathbf{a}_f(k)$, $f \in M_1$, in accordance with (8). The negative product $\mathbf{a}_f(k-1) \cdot r(f)/ \mid \mathcal{N}_f \mid$ is taken separately, and one obtains the stated equation. **q.e.d.**

The diminishing factor $\big(1 - r(f)/\mid \mathcal{N}_f \mid\big)$ appears by definition for all elements of M_1. At subsequent reduction steps, the factor is "transmitted" successively to all probabilities from higher distance levels M_i because any element of M_i has at least one neighbor from M_{i-1}. The main task is now to analyze how this diminishing factor changes, if it is transmitted to the next higher distance level. We denote

$$\sum_{f \notin M_0} \mathbf{a}_f(k) = \sum_{f \notin M_0} \mu(f, v) \cdot \mathbf{a}_f(k-v) + \sum_{f' \in M_0} \mu(f', v) \cdot \mathbf{a}_{f'}(k-v), \tag{11}$$

i.e., the coefficients $\mu(\tilde{f}, v)$ are the factors at probabilities after v steps of an expansion of $\sum_{f \notin M_0} \mathbf{a}_f(k)$. For $f \in M_1$ we have $\mu(f, 1) = 1 - r'(f)/ \mid \mathcal{N}_f \mid$,

and $\mu(f,1)=1$ for the remaining $f \in \mathcal{M}_{\mathbf{s}} \backslash (M_0 \cup M_1)$. For $f' \in M_0$, Lemma 1 implies:

$$\mu(f',1) = \sum_{\substack{i=1 \\ f_i \in M_1 \cap \mathcal{N}_{f'}}}^{s(f')} \frac{\varphi(f_i,f',1)}{|\,\mathcal{N}_{f'}\,|} \tag{12}$$

Starting from step $(k-1)$, the generated probabilities $\mathbf{a}_{f'}(k-u)$ are expanded in the same way as all other probabilities.

Except for the initial values, the coefficients $\mu(f,v)$, $f \notin M_0$, and $\mu(f',v)$, $f' \in M_0$, can be treated in the same way because the expansion (8) is valid for both $\mathbf{a}_f(k)$ and $\mathbf{a}_{f'}(k)$. Therefore, we can apply the same considerations which led to equation (9) to $\mu(f,v)\cdot\mathbf{a}_f(k-v)$ and $\mu(f',v)\cdot\mathbf{a}_{f'}(k-v)$ from (11). We perform an inductive step from $(k-v+1)$ to $(k-v)$ and we suppose that all terms within $\{\,...\,\}$ of (9) are multiplied by the corresponding $\mu(f,v-1)$:

$$\mathbf{a}_f(k-v)\cdot\Bigg\{\mu(f,v-1)\cdot\Bigg(\frac{|\,\mathcal{N}_f\,|-r(f)}{|\,\mathcal{N}_f\,|} - \sum_{\substack{i=1 \\ f_i>f}}^{s(f)} \frac{\big(k+1\big)^{-(\mathcal{Z}(f_i)-\mathcal{Z}(f))}}{|\,\mathcal{N}_f\,|}\Bigg) +$$

$$+ \sum_{\substack{i=1 \\ f_i \geq f}}^{s(f)+q(f)} \frac{\mu(f_i,v-1)}{|\,\mathcal{N}_f\,|}\cdot\frac{1}{\big(k+1\big)^{\mathcal{Z}(f_i)-\mathcal{Z}(f)}} + \sum_{\substack{j=1 \\ f_j<f}}^{r(f)} \frac{\mu(f_j,v-1)}{|\,\mathcal{N}_f\,|}\Bigg\}$$

$$= \mathbf{a}_f(k-v)\cdot\mu(f,v).$$

We obtain for $v \geq 2$:

$$\mu(f,v) = \mu(f,v-1)\cdot D_f(k-v) + \sum_{f' \leq f}\frac{\mu(f',v-1)}{|\,\mathcal{N}_f\,|} + \sum_{f''>f}\frac{\mu(f'',v-1)}{|\,\mathcal{N}_f\,|}\cdot\varphi(f'',f,v).$$

It is important to note that the summands are divided by the same value $|\mathcal{N}_f|$, see (9). We set $\mu(f,j) := 1-\nu(f,j)$ because we are interested in the convergence of both values $\mu(f,j)$ and $\nu(f,j)$ to zero. Substituting $\mu(\tilde{f},j)$ by $1-\nu(\tilde{f},j)$ and rearranging both sides leads to the same structure of the equation as for $\mu(\tilde{f})$:

Lemma 2 *The following recurrent relation is valid for the coefficients $\nu(f,v)$:*

$$\nu(f,v) = \nu(f,v-1)\cdot D_f(k-v) + \sum_{f' \leq f}\frac{\nu(f',v-1)}{|\,\mathcal{N}_f\,|} + \sum_{f''>f}\frac{\nu(f'',v-1)}{|\,\mathcal{N}_f\,|}\cdot\varphi(f'',f,v).$$

Furthermore,

$$\nu(f,i) = \begin{cases} 0, & \text{if } f \in M_j,\ j>i; \\ \dfrac{r(f)}{|\,\mathcal{N}_f\,|}, & \text{if } f \in M_1,\ i=1; \\ 1 - \displaystyle\sum_{\substack{l=1 \\ f_l \in M_1 \cap \mathcal{N}_f}}^{s(f)} \frac{\varphi(f_l,f,1)}{|\,\mathcal{N}_f\,|}, & \text{if } f \in M_0,\ i=1. \end{cases}$$

We note that the representation from Lemma 2 is a linear function with respect to the values $\nu(f,v)$, i.e., once an arithmetic term f has been added (or generated) by the previous value of $\nu(f,v-1)$ or "neighbouring" values $\nu(f',v-1)$, this term is multiplied by factors that only depend either on structural properties of the neighborhood relation (that is the factor $(s(f)+1)/\mid\mathcal{N}_f\mid$ from D_f) or on differences of the objective function which are expressed by $\varphi(f',f,v)$. The differences $\mathcal{Z}(f')-\mathcal{Z}(f)$ do not change during the recursive calculation of values $\nu(f,v)$; the only parameter that changes in $\varphi(f',f,v)$ is v. Therefore, any $\nu(f,v)$ and $\mu(f',v)$ can be expressed by a sum $\sum_i T_i$ of arithmetic terms.

We consider in more details the terms associated with elements of M_0 and M_1. We assume a representation $\mu(f',v-1)=\sum_{u'} T'_{u'}$ and $\nu(f,v-1)=\sum_u T_u$. Since there are no $f''<f'$ for $f'\in M_0$, we obtain:

$$\begin{aligned}\mu(f',v) &= D_{f'}(k-v)\cdot\sum_{u'} T'_{u'}(f') + \sum_{f>f'}\frac{1-\sum_u T_u(f)}{\mid\mathcal{N}_{f'}\mid}\cdot\varphi(f,f',v)\\ &= \sum_{f>f'}\frac{\varphi(f,f',v)}{\mid\mathcal{N}_{f'}\mid} + D_{f'}(k-v)\cdot\sum_{u'} T'_{u'}(f') - \sum_{f>f'}\frac{\sum_u T_u(f)}{\mid\mathcal{N}_{f'}\mid}\cdot\varphi(f,f',v).\end{aligned}$$

As can be seen, the values $\sum_{f>f'}\varphi(f,f',v)/\mid\mathcal{N}_{f'}\mid$ are generated at each time step with the corresponding $\varphi(f,f',v)$. The remaining summands of $\mu(f',v)$ are individual arithmetic terms f_u and $f'_{u'}$ from the previous step, multiplied by a positive or a negative factor.

In the same way we obtain for elements of M_1:

$$\begin{aligned}\nu(f,v) = \frac{r(f)}{\mid\mathcal{N}_f\mid} + \nu(f,v-1)\cdot D_f(k-v) + \sum_{f''>f}\frac{\nu(f'',v-1)}{\mid\mathcal{N}_f\mid}\cdot\varphi(f'',f,v) -\\ -\sum_{\substack{f'<f\\ f'\in M_0}}\frac{\sum_{u'} T'_{u'}(f')}{\mid\mathcal{N}_f\mid},\end{aligned}$$

where $r(f)/\mid\mathcal{N}_f\mid$ is from $(1-(s(f)+1)/\mid\mathcal{N}_f\mid)$. Again, the sign changes only for terms $f'_{u'}(f')$ because $D_f(k-v)>0$. The term $r(f)/\mid\mathcal{N}_f\mid$ appears in all recursive equations of $\nu(f,v)$, $f\in M_1$ and $v\geq 1$, and the same is valid for the value $\sum_{f>f'}\varphi(f,f',v)/\mid\mathcal{N}_{f'}\mid$ in all $\mu(f',v)$, $f'\in M_0$. Therefore, all arithmetic terms f are derived from expressions of the type $r(f)/\mid\mathcal{N}_f\mid$ and $\sum_{f>f'}\varphi(f,f',v)/\mid\mathcal{N}_{f'}\mid$. This justifies the following

Definition 3 *The expressions* $r(f)/\mid\mathcal{N}_f\mid$, $f\in M_1$, *and* $\sum_{f>f'}\varphi(f,f',v)/\mid\mathcal{N}_{f'}\mid$, $f'\in M_0$, *are called* source terms *of* $\nu(f,v)$ *and* $\mu(f',v)$, *respectively.*

During an expansion of $\sum_{f\notin M_0}\mathbf{a}_f(k)$ backwards according to (8), the source terms are distributed permanently to higher distance levels M_j. That means, in the same way as for M_1, the calculation of $\nu(f,v)$ is repeated almost identically at any step, only the "history" of generations becomes longer. Therefore, at

higher distance levels the notion of a source term can be defined by an inductive step:

Definition 4 *If $\tilde{f} \in M_i$, $i > 1$, any term that is generated according to the equation of Lemma 2 from a source term of $\nu(f, v-1)$, where $f \in M_{i-1} \cap \mathcal{N}_{\tilde{f}}$, is said to be a source term of $\nu(\tilde{f}, v)$.*

We introduce a counter $\mathbf{r}(T)$ to terms f that indicates the step at which the term has been generated from source terms. The value $\mathbf{r}(T)$ is called the rate of a term and we set $\mathbf{r}(T) = 1$ for source terms f.

The value $\mathbf{r}(T) > 1$ is assigned to terms related to M_0 and M_1 in a slightly different way compared to higher distance levels because at the first step the $f' \in M_0$ do not participate in the expansion of $\sum_{f \notin M_0} \mathbf{a}_f(k)$. Furthermore, in the case of M_0 and M_1 we have to take into account the changing signs of terms that result from the simultaneous consideration of $\nu(f, v)$ (for M_1) and $\mu(f', v)$ (for M_0). Basically, a rate $\mathbf{r}(T) > 1$ is assigned to a term that was at the preceding step an $(\mathbf{r}(T)-1)$-term or $(\mathbf{r}(T)-2)$-term at the same or a higher distance level, respectively, or an $\mathbf{r}(T)$-term at a lower distance level.

Definition 5 *A term f' is called a j^{th}* rate term *of $\mu(f', v)$, $f' \in M_0$ and $j \geq 2$, if either $f' = -T$ and $\mathbf{r}(T) = j-1$ for some $\nu(f, v-1)$, $f \in M_1 \cap \mathcal{N}_{f'}$, or $\mathbf{r}(T') = j-1$ for some $\mu(f'', v-1)$, $f'' \in M_0 \cap \mathcal{N}_{f'}$.*

A term f is called a j^{th} rate term *of $\nu(f, v)$, $f \in M_1$ and $j \geq 2$, if $\mathbf{r}(T) = j-2$ for some $\nu(\tilde{f}, v-1)$, $\tilde{f} \in M_2 \cap \mathcal{N}_f$, $\mathbf{r}(T) = j-1$ for some $\nu(\tilde{f}, v-1)$, $\tilde{f} \in M_1 \cap \mathcal{N}_f$, or $f = -T'$ and $\mathbf{r}(T') = j-1$ for some $f' \in M_0 \cap \mathcal{N}_f$ with respect to $\mu(f', v-1)$.*

A term f is called a j^{th} rate term *of $\nu(\tilde{f}, v)$, $\tilde{f} \in M_i$ and $i, j \geq 2$, if $\mathbf{r}(T) = j-1$ for some $\nu(\tilde{f}', v-1)$, $\tilde{f}' \in M_{i+1} \cap \mathcal{N}_{\tilde{f}}$, $\mathbf{r}(T) = j-1$ for some $\nu(\tilde{f}', v-1)$, $\tilde{f}' \in M_i \cap \mathcal{N}_{\tilde{f}}$, or f is a j^{th} rate term of $\nu(f, v-1)$ for some $f \in M_{i-1}$.*

The classification of terms will be used for a partition of the summation over all terms which constitute particular values $\nu(f, v)$ and $\mu(f', v)$. Let $\mathcal{T}_j(\tilde{f}, v)$ be the set of j^{th} rate arithmetic terms of $\nu(f, v)$ $(\mu(f', v))$ related to $\tilde{f} \in \mathcal{M}_{\mathbf{s}}$. We set

$$\mathbf{S}_j(\tilde{f}, v) := \sum_{f \in \mathcal{T}_j(\tilde{f}, v)} T. \tag{13}$$

The same notation is used in case of $\tilde{f} = f' \in M_0$ with respect to $\mu(f', v)$. Now, the coefficients $\nu(\tilde{f}, v)$, $\mu(f', v)$, can be represented by

$$\nu(\tilde{f}, v) = \sum_{j=1}^{v-i+1} \mathbf{S}_j(\tilde{f}, v) \quad \text{and} \quad \mu(f', v) = \sum_{j=1}^{v} \mathbf{S}_j(f', v). \tag{14}$$

We compare the computation of $\nu(f, v)$ (and $\mu(f', v)$) for two different values $v = k_1$ and $v = k_2$, i.e., $\nu(f, v)$ is calculated backwards from k_1 and k_2, respectively. Let $\mathbf{S}_j^1$ and $\mathbf{S}_j^2$ denote the corresponding sums of terms related to two different starting steps k_1 and k_2, respectively. Since the source terms depend on v only,

see Definition 3, we obtain from the simple relation $k_2 - (k_2 - k_1 + v) = k_1 - v$ the following equality:

Lemma 3 *Given $k_2 \geq k_1$ and $1 \leq j \leq k_1$, then*

$$\mathbf{S}_j^1(f, v) = \mathbf{S}_j^2(f, k_2 - k_1 + v).$$

We recall that our main goal is to upper bound the sum $\sum_{f \notin M_0} \mathbf{a}_f(k)$. When $\mathbf{a}(0)$ denotes the initial probability distribution, we can derive

$$(15)\quad \Big|\sum_{f\notin M_0} \mathbf{a}_f(k_2) - \sum_{f\notin M_0} \mathbf{a}_f(k_1)\Big| \leq \sum_{f\notin M_0} \big|\big(\nu(f,k_2) - \nu(f,k_1)\big)\cdot \mathbf{a}_f(0)\big| + \\ + \sum_{f'\in M_0} \big|\big(\mu(f',k_2) - \mu(f',k_1)\big)\cdot \mathbf{a}_{f'}(0)\big|\,.$$

Any element of the initial distribution $\mathbf{a}_f(0)$ is simply replaced by 1 and we obtain for the first part of the sum in accordance with Lemma 3:

$$(16)\qquad \sum_{f\notin M_0} \big|\big(\nu(f,k_2) - \nu(f,k_1)\big)\cdot \mathbf{a}_f(0)\big| \leq \sum_{f\notin M_0} \sum_{j=k_1+1}^{k_2} |\mathbf{S}_j^2(f,k_2)|\,.$$

For $\tilde{f} \in M_i \neq M_1$, M_0 and $j \geq 2$ we obtain immediately from Lemma 2 and Definition 5:

$$(17)\quad \mathbf{S}_j(\tilde{f}, v) = \mathbf{S}_{j-1}(\tilde{f}, v-1)\cdot D_{\tilde{f}}(k-v) + \sum_{\substack{\tilde{f}' < \tilde{f} \\ \tilde{f}' \in M_{i+1}}} \frac{\mathbf{S}_{j-2}(\tilde{f}', v-1)}{|\,\mathcal{N}_{\tilde{f}}\,|} + \\ + \sum_{\substack{\tilde{f}'' > \tilde{f} \\ \tilde{f}'' \in M_{i+1}}} \frac{\mathbf{S}_{j-2}(\tilde{f}'', v-1)}{|\,\mathcal{N}_{\tilde{f}}\,|}\cdot\varphi(\tilde{f}'', \tilde{f}, v) + \sum_{\substack{\tilde{f}' < \tilde{f} \\ \tilde{f}' \in M_{i-1}}} \frac{\mathbf{S}_j(\tilde{f}', v-1)}{|\,\mathcal{N}_{\tilde{f}}\,|} + \\ + \sum_{\substack{\tilde{f}' < \tilde{f} \\ \tilde{f}' \in M_{i-1}}} \frac{\mathbf{S}_j(\tilde{f}', v-1)}{|\,\mathcal{N}_{\tilde{f}}\,|} + \sum_{\substack{\tilde{f}'' > \tilde{f} \\ \tilde{f}'' \in M_{i-1}}} \frac{\mathbf{S}_j(\tilde{f}'', v-1)}{|\,\mathcal{N}_{\tilde{f}}\,|}\cdot\varphi(\tilde{f}'', \tilde{f}, v).$$

In case of $f \in M_1$ and $j \geq 2$, we have in accordance with Definition 5:

$$(18)\quad \mathbf{S}_j(f, v) = \mathbf{S}_{j-1}(f, v-1)\cdot D_f(k-v) + \\ + \sum_{\substack{\tilde{f}'' > f \\ \tilde{f}'' \in M_2}} \frac{\mathbf{S}_{j-2}(\tilde{f}'', v-1)}{|\,\mathcal{N}_f\,|}\cdot\varphi(\tilde{f}'', t, v) - \sum_{f'\in M_0\cap\mathcal{N}_f} \frac{\mathbf{S}_{j-1}(f', v-1)}{|\,\mathcal{N}_f\,|}.$$

Finally, the corresponding relation for $f' \in M_0$ is given by

$$(19)\quad \mathbf{S}_j(f', v) = \mathbf{S}_{j-1}(f', v-1)\cdot D_{f'}(k-v) \;-\; \sum_{f > f'} \frac{\mathbf{S}_{j-1}(f, v-1)}{|\,\mathcal{N}_{f'}\,|}\cdot\varphi(f, f', v).$$

The recursive application of (17) till (19) generates a tree-like structure $\mathbf{T}(f, j, v)$, where a particular $\mathbf{S}_j(f, v)$ represents the root of the tree. The leaves are source terms from M_1 and M_0 after a full recursion of v steps.

Positive and negative terms are considered separately, and a uniform upper bound of $|\mathbf{S}_j(f, k_2)|$ is calculated from the leaves to the root. Here, we distinguish between M_1 and M_0. The terms from M_0 are multiplied by an extra factor $\varphi(f_1, f, 1)/|\mathcal{N}_f|$, and therefore it is sufficient to consider the terms generated by elements of M_1.

Since any element of M_1 has a neighbour from M_0 and the terms arising from the lowest level are treated separately, the terms from M_0 (i.e., generated by f itself and neighbours $\mathcal{N}_f \cap M_0$) are multiplied by a factor smaller than 1. At each step towards the root, we take the values from elements of M_1 as a new summand to the terms generated at higher levels M_i, $i > 1$. Therefore, the already existing summands are also multiplied by a factor smaller than 1, when summands from neighbouring elements with the same generation step are taken together. The factor is determined by $D' := D_f - (s_{i-1} + r_{i-1})/n$, when n denotes for short an upper bound for $|\mathcal{N}_f|$.

Hence, the analysis of terms can be reduced to the terms (new summands) generated at level M_0. The particular products (summands) can be classified by the number a of self-transitions (i.e., factors $D_f(k - v)$), the number b of transitions to higher levels (factors $\varphi(f', f, v)$), and c steps to the same level or lower levels of the objective function. Thus, we have $a + b + c = v$ for the three types of transitions. For fixed a and b, the products can be upper bounded by

$$(20)\quad \binom{v}{a} \cdot \binom{v-a}{2\cdot b} \cdot \binom{2\cdot b}{b} \cdot \max_{i,j} \prod_{i=1}^{b} \Big(\frac{t_i}{(n)^2} \cdot \frac{1}{(k+2-u_i)^{\omega}} \Big) \cdot \prod_{j=1}^{a} D'_{f_j}(k - u_j),$$

where $t_i \le (n-1)^2$ and ω denotes the uniform difference between neighbouring elements (see the remark after Theorem 1). We employ the STIRLING formula for the following estimation:

$$(21)\quad \prod_{u=1}^{b} \varphi(f_u, f'_u, v_u) \le \prod_{u=1}^{b} \frac{1}{\big(k+2-v_u\big)^{\omega}} \;\le\; \frac{1}{\big(b!\big)^{\omega}} \;\le\; \Big(\frac{e}{b}\Big)^{b\cdot\omega}.$$

The product (20) is analysed for both cases $a \le v/2$ and $a > v/2$ separately. In the first case, $v \ge 2^{O(\Gamma)}$ is sufficient to obtain an exponentially small bound for (20). In the second case, we need $k \ge n^{O(\Gamma)}$ for the factors $\varphi(f_u, f'_u, v_u)$ to be sufficiently small in order to compensate the binomial coefficients.

The analysis of 20) leads to a uniform upper bound of $|\mathbf{S}_j(f, k_2)|$ and we obtain

Lemma 4 *Given the maximum escape depth Γ, then $n^{O(n)} \ge k_2 \ge k_1 > n^{O(\Gamma)}$ implies*

$$\Big|\sum_{f \notin M_0} \big(\nu(f, k_2) - \nu(f, k_1)\big) \cdot \mathbf{a}_f(0)\Big| < k_2^{O(1)} \cdot 2^{-k_1^{O(1-1/\Gamma)}}.$$

In the same way, the corresponding upper bound for $\big(\mu(f', k_1) - \mu(f', k_2)\big)$ is derived.

By our assumption about the size of $\mathcal{F}$ (see Section 2), a value of $k_2 < n^{O(n)}$ allows a complete search of $\mathcal{F}$. We assume that such an upper bound is sufficient for a convergence according to Theorem 1. Now, we can immediately prove

Theorem 2 *When the convergence is guaranteed for $k_2 \leq n^{O(n)}$, then $k > n^{O(\Gamma)} + \log^{O(1)} 3/\varepsilon$ implies for arbitrary initial probability distributions $\mathbf{a}(0)$ and $\varepsilon > 0$ the relation*

$$\sum_{\tilde{f} \notin M_0} \mathbf{a}_{\tilde{f}}(k) < \varepsilon \quad \textit{and therefore,} \quad \sum_{f' \in M_0} \mathbf{a}_{f'}(k) \geq 1 - \varepsilon.$$

Sketch of Proof: We use the following representation:

$$\begin{aligned}\sum_{\tilde{f} \notin M_0} \mathbf{a}_{\tilde{f}}(k) &= \sum_{\tilde{f} \notin M_0} \big(\mathbf{a}_{\tilde{f}}(k) - \mathbf{a}_{\tilde{f}}(k_2)\big) + \sum_{\tilde{f} \notin M_0} \mathbf{a}_{\tilde{f}}(k_2) \\ &= \sum_{\tilde{f} \notin M_0} \big(\nu(\tilde{f}, k_2) - \nu(\tilde{f}, k)\big) \cdot \mathbf{a}_{\tilde{f}}(0) + \\ &\quad + \sum_{f' \in M_0} \big(\mu(f', k) - \mu(f', k_2)\big) \cdot \mathbf{a}_{f'}(0) + \sum_{\tilde{f} \notin M_0} \mathbf{a}_{\tilde{f}}(k_2).\end{aligned}$$

The value k_2 from Lemma 4 is larger but independent of $k_1 = k$, i.e., we can take a $k_2 > k$ such that

$$\sum_{\tilde{f} \notin M_0} \mathbf{a}_{\tilde{f}}(k_2) < \frac{\varepsilon}{3}.$$

We employ Theorem 1, i.e., if the constant Γ from (5) is sufficiently large, the inhomogeneous simulated annealing procedure defined by (1), (2), and (3) tends to the global minimum of $\mathcal{Z}$ on $\mathcal{F}$. We obtain the stated inequality, if additionally both differences $\sum_{\tilde{f} \notin M_0} \big(\nu(\tilde{f}, k_2) - \nu(\tilde{f}, k)\big)$ and $\sum_{f' \in M_0} \big(\mu(f', k) - \mu(f', k_2)\big)$ are smaller than $\varepsilon/3$. Lemma 4 implies that the condition on the differences is satisfied in case of

$$2^{-k^{1/\gamma}} < \frac{\varepsilon}{3},$$

which is valid, if $\log^{\gamma}(3/\varepsilon) < k$. **q.e.d.**

4 Concluding Remarks

We analysed the convergence of logarithmic cooling schedules in terms of the underlying energy landscape. Under natural assumption about the configuration space we succeeded to prove a lower bound on the number of cooling steps that are sufficient to approach with with probability $1-\varepsilon$ the set of optimum solutions. The lower bound is given by $n^{O(\Gamma)} + \log^{O(1)}(1/\varepsilon)$, where Γ is the maximum escape depth from local minima and n describes the size of the neighbourhood of configurations. When Γ can be calculated or estimated, the result can provide good or improved upper bounds for the time complexity of stochastic algorithms.

References

1. S. Azencott (editor). *Simulated Annealing: Parallelization Techniques.* Wiley & Sons, New York, 1992.
2. E.H.L. Aarts and J.H.M. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach*, Wiley & Sons, New York, 1989.
3. A. Albrecht, S.K. Cheung, K.S. Leung, and C.K. Wong. Stochastic Simulations of Two-Dimensional Composite Packings. *J. of Computational Physics*, 136(2):559 – 579, 1997.
4. O. Catoni. Rough Large Deviation Estimates for Simulated Annealing: Applications to Exponential Schedules. *Annals of Probability*, 20(3):1109 – 1146, 1992.
5. O. Catoni. Metropolis, Simulated Annealing, and Iterated Energy Transformation Algorithms: Theory and Experiments. *J. of Complexity*, 12(4):595 – 623, 1996.
6. V. Černy. A Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. Preprint, Inst. of Physics and Biophysics, Comenius Univ., Bratislava, 1982 (see also: *J. Optim. Theory Appl.*, 45:41 – 51, 1985).
7. B. Hajek. Cooling Schedules for Optimal Annealing. *Mathem. Oper. Res.*, 13:311 – 329, 1988.
8. M. Kearns, M. Li, L. Pitt, and L.G. Valiant. Recent Results on Boolean Concept Learning. In *Proc. 4^{th} Int. Workshop on Machine Learning*, pp. 337 – 352, 1987.
9. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671 – 680, 1983.
10. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *J. of Chemical Physics*, 21(6):1087 – 1092, 1953.
11. S. Rajasekaran and J.H. Reif. Nested Annealing: A Provable Improvement to Simulated Annealing. *J. of Theoretical Computer Science*, 99(1):157-176, 1992.
12. F. Romeo and A. Sangiovanni-Vincentelli. A Theoretical Framework for Simulated Annealing. *Algorithmica*, vol. 6, no. 3, pp. 302 – 345, 1991.
13. E. Seneta. *Non-negative Matrices and Markov Chains.* Springer-Verlag, New York, 1981.
14. A. Sinclair and M. Jerrum. Approximate Counting, Uniform Generation, and Rapidly Mixing Markov Chains. *Information and Computation*, 82:93 – 133, 1989.
15. A. Sinclair and M. Jerrum. Polynomial-Time Approximation Algorithms for the Ising Model. *SIAM J. Comput.*, 22(5):1087 – 1116, 1993.
16. G. Sorkin. Efficient Simulated Annealing on Fractal Energy Landscapes. *Algorithmica*, 6:367-418, 1991.
17. K. Verbeurgt. Learning DNF under the Uniform Distribution in Quasi-Polynomial Time. In *Proc. of the 3^{rd} Annual Workshop on Computational Learning Theory*, pp. 314 – 326, 1990.

Hierarchical State Machines

Mihalis Yannakakis

Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974

Abstract. Hierarchical state machines are finite state machines whose states themselves can be other state machines. Hierarchy is a useful construct in many modeling formalisms and tools for software design, requirements and testing. We summarize recent work on hierarchical state machines with or without concurrency. We discuss issues of expressiveness and succinctness, the complexity of basic operations (such as emptiness, equivalence etc.), and algorithmic problems in the analysis of hierarchical machines to check for their properties.

1 Introduction

Finite state machines constitute one of the most fundamental modeling mechanisms in Computer Science. They have been widely used to model systems in a wide variety of areas, including sequential circuits, event-driven software, communication protocols and many others. In its most basic form, a finite state machine consists of a directed graph whose nodes represent system states and edges correspond to system transitions. When equipped with an input (and/or output) alphabet, FSM's become language recognizers defining the regular languages (or transducers). A rich theory of FSM's had been developed since the 50's regarding their expressive power, their operations and the analysis of their properties.

In practice, several extensions are useful to enhance the expressive power and to describe more complex systems more efficiently. Two popular extensions that are fairly well understood are the addition of variables to form so-called *extended finite state machines*, and the addition of concurrency (communication) to form *communicating finite state machines*. A third useful extension that has been less well studied is the addition of a hierachical (nesting) capability, to form *hierarchical state machines*, i.e., machines whose nodes can be ordinary states or superstates which are FSM's themselves. This is a very useful construct to structure and represent large systems.

The notion of hierarchical FSMs was popularized by the introduction of Statecharts [Har87], and exists in many related specification formalisms such as Modecharts [JM87] and Rsml [LHHR94]. It is a central component of various object-oriented software development methodologies developed in recent years, such as Omt [RBPE91], Room [SGW94], and the Unified Modeling Language (Uml [BJR97]). This capability is commonly available also in commercial computer-aided software engineering tools that are coming out, such as

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 315–330, 2000.
© Springer-Verlag Berlin Heidelberg 2000

STATEMATE (by i-Logix), OBJECTIME DEVELOPER (by ObjecTime), and RATIONALROSE (by Rational) (the last two are now combined into RationalRose-RealTime).

The nesting capability is useful also in formalisms and tools for the requirements and testing phases of the software development cycle. On the requirements side, it is used to specify scenarios (or *use cases* [Jac92]) in a structured manner. For instance, the new ITU standard Z.120 (MSC'96) for message sequence charts [RGG96] formalizes scenarios of distributed systems in terms of hierarchical graphs built from basic MSC's. The Lucent UBET toolset for behavioral requirements engineering uses a similar formalism and models requirements by hierarchical message sequence charts (UBET stands for Lucent Behavioral Engineering Toolset and is based on the MSC/POGA prototype tools [AHP96, HPR97]).

On the testing side, FSMs are used often to model systems for the purpose of test generation, and again the nesting capability is useful to model large systems. For example, Teradyne's commercial tool TESTMASTER [Apf95] is based on an extended hierarchical FSM model, i.e. it employs hierarchy and variables (but not concurrency).

In this paper we will summarize recent work on the impact of adding hierarchy to FSM, when added alone or in conjunction with concurrency. Most of the results come from the papers [AY98,AKY99,AY99] to which we refer for more details. In Section 2 we discuss hierarchical state machines: their expressive power and succinctness as compared to ordinary FSM's, the effect of nondeterminism vs. determinism in this context, the complexity of basic operations such as checking for emptiness, universality, equivalence etc, and the model checking of properties on hierarchical state machines. In Section 3 we discuss the effect of combining concurrency with hierarchy and address similar questions. In Section 4 we discuss hierarchical message sequence charts. In Section 5 we comment briefly on issues of testing, and we conclude in Section 6.

2 Hierarchical FSMs

There are many variants in definitions of finite state machines, depending on whether one views them as language acceptors/generators (automata), or machines that react with their environment and produce output (eg. Mealy and Moore machines, I/O automata), or logical models (Kripke structures) etc. For concreteness, let's settle with the automata definition. An ordinary *basic finite state machine* M consists of a finite set Q of states, an initial (or entry) state q_0, a set of final (or exit states) F, a finite input alphabet Σ, and a set E of transitions, $E \subseteq Q \times \Sigma \times Q$. Finite state machines are usually drawn as directed graphs with the states as nodes and the transitions as edges labeled by the inputs. The language $L(M)$ accepted by M is the set of strings over Σ that label paths from the initial state to a final state.

Hierarchical machines (HSM) are finite state machines whose states are themselves other basic FSM's or previously defined hierarchical machines. For sim-

plicity we just give the formal definition for the single entry-single exit case. Formally, HSM's are defined inductively as follows. In the base case, a basic FSM is a hierarchical machine. Inductively, suppose that $\mathcal{M}$ is a set of HSM's. If N is a FSM with set of states Q and μ is a labeling function $\mu : Q \mapsto \mathcal{M}$ that associates each state $q \in Q$ with a HSM in $\mathcal{M}$ then the triple $(N, \mathcal{M}, \mu)$ is a HSM.

A hierarchical machine H represents in a compact notation a corresponding ordinary FSM $flat(H)$, the flattened version of H. The flat FSM can be constructed inductively as follows. In the base case, $flat(H) = H$. For the inductive step, if $H = (N, \mathcal{M}, \mu)$, then replace in the FSM N each state q by a copy of the flat version $flat(\mu(q))$ of its corresponding machine; note: if several states map to the same machine of $\mathcal{M}$ then we replace them by distinct copies of the flat machines with distinct states. Transitions of N coming into state q are directed to the entry state of the copy of $flat(\mu(q))$ and transitions of N coming out of q are now coming out of the exit state of $flat(\mu(q))$. If the entry state of N is q_0 and the exit state is q_f, then the entry state of $flat(H)$ is the entry state of $flat(\mu(q_0))$ and the exit state of $flat(H)$ is the exit state of $flat(\mu(q_f))$.

The definition of hierachical machines for the multiple entry - multiple exit case is similar, except that one has to specify in this case the set of entry and exit states, and to specify for every transition (q, a, q') of the top level FSM N in addition an associated exit state of $\mu(q)$ and an entry state of $\mu(q')$. We omit here the formalization.

An HSM H can be represented by a rooted DAG, whose leaves are basic FSM's and each internal node u has an associated FSM M_u and a mapping μ_u from the states of M_u to the children of u. The size of the (representation of the) HSM H is the sum of the sizes (numbers of states and transitions) of all the machines M_u at the nodes of the DAG.

Note that the same machine M_j may be used by many other machines M_i and by many of their states. This reuse permits exponential succinctness as compared to the flattened machine. Consider for example the HSM depicted in Figure 1, over a unary alphabet $\Sigma = \{a\}$. At each level, each M_i has two states that are mapped to the machine M_{i-1} at the previous level. The size of the HSM in this example is $O(n)$. However, the flattened machine in this case is simply a path of length $2^n - 1$. Thus, the machine is a counter that accepts just one string, the string of length $2^n - 1$.

2.1 Expressibility

Hierarchical state machines are essentially pushdown machines with a pushdown stack that is bounded by the input. They capture of course the same languages as ordinary FSM (regular languages), they gain only in succinctness. If a HSM H is defined by a DAG D of r machines $\{M_i\}$, each with at most n states, and if the nesting depth is m (i.e. the maximum length of a path in the DAG), then the flattened FSM has at most $O(n^m)$ states: every state of $flat(H)$ can be represented as a tuple $(q_1, q_2, ...q_t)$, $t \leq m$, of states of the given machines M_i, where q_1 is a state of the top-level machine, q_2 is a state of $\mu(q_1)$, and so

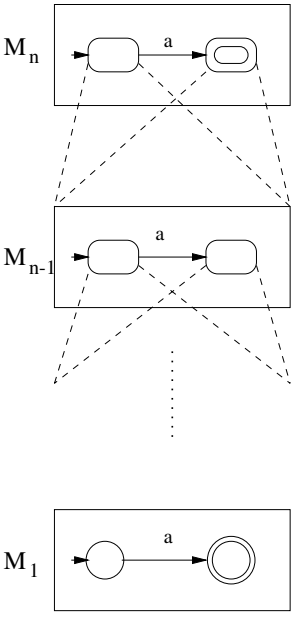


Fig. 1. An exponential counter

forth, i.e. every other state q_j of the tuple is a state of the machine to which the previous state q_{j-1} of the tuple is mapped. Thus, hierarchical machines can be translated to ordinary FSM's at an exponential cost. This is in general inherent.

We can consider the impact of hierarchy on the relation between determinism and nondeterminism. Recall that a FSM is deterministic if for every state q and every input symbol a there is at most one transition out of q labeled a. We say that a HSM is deterministic if its flat FSM is deterministic.

In the case of ordinary FSM's there is of course an exponential gap in succinctness between deterministic and nondeterministic FSM's. For hierarchical machines the gap becomes doubly exponential. That is, there are (nondeterministic) HSM's of size n such that the smallest equivalent deterministic HSM has size doubly exponential in n. An example of such a family of languages is the set L_n of strings w of length $2 \cdot 2^n$ such that there is an index i for which $w_i = w_{i+2^n}$. A nondeterministic HSM of linear size can guess i, record w_i, count 2^n as in Figure 1, and check that $w_i = w_{i+2^n}$. On the other hand, deterministic HSM's need doubly exponential size for this language [AKY99].

Of course, the gap between nondeterministic HSM's and deterministic HMS or even basic FSM's is no worse than doubly exponential: one exponential suffices to translate from HSM to nondeterministic basic FSM, and another exponential to deterministic FSM.

Comparing deterministic hierarchical machines with nondeterministic basic FSM's, there is an exponential gap in both directions. That is, one the one hand, there is a family of languages that is accepted by linear size nondeterministic

FSM's but needs exponential size for deterministic HSM's; one such language is the set of strings w of length $2n$ such that $w_i = w_{i+n}$ for some i. On the other hand, there is a language that can be accepted by a linear size deterministic HSM but needs exponential size for nondeterministic FSM's; one such language is the set of strings of the form $w\#w^R$, i.e. the second half w^R is the reverse of the first half w of the input string.

2.2 Operations

Usual operations of interest for a state machine and its language L are emptiness (is $L = \emptyset$?), universality (is $L = \Sigma^*$), and complementation (construct a machine that accepts the complement). For pairs of state machines M_1, M_2, operations of interest include intersection (is $L(M_1) \cap L(M_2) = \emptyset$?, and compute a machine that accepts the intersection), and comparison of the two machines: equivalence (is $L(M_1) = L(M_2)$?) and containment or inclusion (is $L(M_1) \subseteq L(M_2)$?) We discuss the complexity of these operations for hierarchical machines.

Emptiness. This is equivalent to a reachability problem in the flattened machine: can the initial state reach a final state? This problem can be solved in linear time in the size of the HSM by a simple dynamic programming search algorithm. In the case of flat FSM's the problem is in NL (nondeterministic logspace). In the case of hierarchical machines, the problem becomes P-complete.

Universality. As in the case of basic FSM's, this is a harder problem. As is well-known, for basic FSM's universality is PSPACE-complete. Hierarchy costs an additional exponential in this case: universality for HSM's is EXPSPACE-complete. Note that universality is emptiness for the complement, and emptiness is easy (hence complementation is hard). For deterministic HSM's we can complement them easily, and hence we can solve universality in linear time.

Intersection. For basic FSM's intersection is a polynomial operation. This holds also for intersection of a HSM with a FSM: that is, given a HSM H and a (basic) FSM M, we can define their product, which is another HSM H' such that $L(H') = L(H) \cap L(M)$. The size of H' is bounded by $|H||M|^2$; basically, the product involves at worst pairing every machine in the representation of H with a copy of M initialized at each state of M. However there is no such simple product construction for two HSM's: determining whether the intersection of the languages of two HSM's is empty turns out to be PSPACE-complete.

Equivalence and Inclusion. Since universality is EXPSPACE-hard, it follows that equivalence and containment are at least as hard. Indeed they are both in EXPSPACE, by flattening and complementing one of the machines, and forming the product with the other machine. In the case of deterministic HSM's (for which universality as well as complementation is easy), inclusion turns out to be PSPACE-complete. As for equivalence of deterministic HSM's, it can be done in PSPACE but the precise complexity is open.

2.3 Verification

A hierarchical state machine H is considered now as modeling a system. The model checking problem asks whether a given system model satisfies a given property P. The model checking problem has been studied for a variety of classes of properties, and there are several software tools to automatically perform the verification for finite state systems (eg. SPIN [Hol97], COSPAN [Kur94], SMV [McM93]). Typically, properties are defined either using automata, or some form of logic (linear time or branching time). Usually models are finite state structures that have labelled nodes instead of edges; specifically, there is a finite set of propositions, and the nodes (states) are labelled by the propositions satisfied at the state. We defined above hierarchical machines using edge labels (to conform with language generation), but one can equally well define them using node labels; there is not a significant difference between edge-labelled and node-labelled machines. We say that the hierarchical machine H satisfies a property P if $flat(H)$ does.

The simplest kind of property is a state invariant, i.e., the property that starting from the initial state, the system stays within a specified subset of states. This amounts to a simple reachability problem and can be solved in linear time in the size of the HSM. So-called 'safety' properties that depend on finite computations can be also reduced to a reachability problem.

A much richer class of properties is the class of linear time properties expressed by linear temporal logic (LTL) [Pnu77] or by Buchi automata. These are properties on the infinite computations of the system (i.e. paths of the state machine). The model satisfies a given property if all of its paths starting from the initial state satisfy the property. A Buchi automaton A is like a usual finite string automaton (i.e. a basic FSM); the only difference is that its language is defined to be a set of infinite strings, namely the strings that label infinite paths (starting from the initial state) which go infinitely often through one of the accepting states. We will not define LTL here, but suffice it to say that every LTL formula ϕ can be translated to an equivalent Buchi automaton A of size $O(2^{|\phi|})$ [VW86]; and in the automata theoretic approach (and in tools like SPIN) this is the way that LTL properties are checked, i.e. by conversion to automata. This does not hurt the complexity, because the exponential dependence on the size of the formula is unavoidable (but formulas are typically quite small, so this is tolerable). It is more convenient to use a Buchi automaton to specify the *bad* computations, those that signify an error, rather than the correct computations. Then the model checking problem for a model M amounts to the question of nonemptiness of the intersection $L(M) \cap L(A)$, where the model M is now viewed as generating a set of infinite strings.

The core algorithmic problem in this case is not just reachability, but the following *Cycle Detection problem*: Given a directed graph (a basic FSM) with an initial node and a set of specified 'special' nodes, can the initial node reach a cycle that contains one of the special nodes? The model checking problem for a flat FSM M and a Buchi automaton A is reduced to this cycle detection problem, by simply taking the product of M and A and letting the special nodes be those

that correspond to accepting nodes of A. The cycle detection problem can be solved in linear time, either by computing the strongly connected components of the graph, or by an alternative "nested depth-first search" algorithm that is useful for on-the-fly verification [CVWY92].

The model checking problem for hierarchical machines H can be similarly reduced to the cycle detection problem for HSM's: As we mentioned earlier, we can form the product of the HSM H with the automaton A, which is another HSM H'. Then H and A have a nonempty intersection iff $flat(H')$ has a reachable cycle that contains a special node. Finding strong components of $flat(H')$ is not convenient (there is too many of them), but the cycle detection problem can still be solved in linear time in the size of H' by suitable adaptation of the nested depth first search algorithm, see [AY98]. This yields an algorithm for model checking of an HSM H with a Buchi automaton A that is linear in the size of the HSM and quadratic in the size of A (usually the system is much larger than the property, so the dependence on the system size is the more critical measure). Thus, we can verify a herarchical machine without flattening it, i.e., there is essentially no penalty for the exponential succinctness of hierarchy in this case.

The other major brand of temporal logic is branching temporal logic. In the case of basic FSM's, checking a model M for a formula ϕ in branching time logic CTL is easier than LTL: it can be done in time $O(|M||\phi|)$, i.e. linear in both the size of the model and the specification. In the case of hierarchical systems, if we flatten the hierarchical machine and apply the pure FSM algorithm, the complexity of the algorithm will be exponential in the size of the hierarchical model and linear in the size of the formula $|\phi|$. Typically, formulas are small and models are large, so this is not a good trade off. There is an alternative better algorithm, which makes the trade off in the other way, and has complexity exponential in the size of the formula. The dependence on the model size is affected by the number d of exit nodes allowed in the nested machines. Namely, the model checking problem for a HSM H and a CTL formula ϕ can be solved in time $O(|H|2^{|\phi|d})$, and also in polynomial space. Thus, in general we can again avoid flattening the HSM. In the single exit case, the time complexity is linear in the size of the HSM, though in the multiple exit case it grows exponentially with the number of exit states. These dependencies are probably inherent: In the single-exit case, if the formula is part of the input then the problem is PSPACE-complete, so the complexity presumably has to be exponential in either the model size or the formula size, and of course we are better off having the exponential dependence on the formula rather than the model. In the multiple exit case, there is a fixed formula for which the problem is PSPACE-complete (so the model size has to enter exponentially somehow). We summarize in the table of Figure 2 the complexity of the various analysis problems in the checking of properties for HSM's, as compared to FSM's.

	Ordinary FSM's	Hierarchical FSM's
Reachability	$O(\|M\|)$	$O(\|M\|)$
Automata-checking	$O(\|M\| \cdot \|A\|)$	$O(\|M\| \cdot \|A\|^2)$
LTL model checking	$O(\|M\| \cdot 2^{\|\phi\|})$	$O(\|M\| \cdot 4^{\|\phi\|})$
CTL model checking	$O(\|M\| \cdot \|\phi\|)$	$O(\|M\| \cdot 2^{\|\phi\| d})$

Fig. 2. Summary of FSM and HSM model checking

3 Concurrency and Hierarchy

A *concurrent (or communicating) hierarchical state machine*, CHM, combines concurrency and hierarchy in an arbitrarily nested manner. We define it formally here for the single entry -single exit case; the definition for the multiple entry/exit case is similar.

A CHM is defined again inductively. In the base case, every ordinary FSM is a CHM. For the induction step, a CHM H is either a hierarchical combination of previously defined CHM's exactly as before, or it is a concurrent (parallel) combination $M_1 \parallel M_2 \parallel \cdots \parallel M_k$.

The flat basic FSM, $flat(H)$ is defined again inductively. It is the same as before in the basis case and in the case of a hierarchical combination. In the concurrent combination, $flat(H)$ is the product of the FSM's $flat(M_i)$, defined as follows. Assume that $flat(M_i)$ has set of states Q_i, alphabet Σ_i, initial state q_0^i, final state q_f^i, and transition relation E_i. Then the state space of H is $Q_1 \times \cdots \times Q_k$, the alphabet is $\Sigma_1 \cup \cdots \cup \Sigma_k$, the initial state is $(q_0^1, \cdots, q_0^k)$, the final state is $(q_f^1, \cdots, q_f^k)$, and the transition relation has a transition labelled a from state $(u_1, \cdots, u_k)$ to state $(v_1, \cdots, v_k)$ if every component machine M_i, whose alphabet Σ_i contains the symbol a, has a transition from u_i to v_i labelled a. The language $L(H)$ of a CHM H is the language of its corresponding FSM $flat(H)$.

A concurrent hierarchical machine H can be represented by a rooted DAG (directed acyclic graph) corresponding to the way it is built from basic FSM's. The leaves are basic FSM's and each internal node is either labelled by the concurrent combinator $\parallel$, or corresponds to a hierarchical combination and is labelled by a machine N and a mapping μ from the states of N to the children of the node. Since $\parallel$ is an associative operator, we can assume that the children of $\parallel$ nodes are not themselves $\parallel$ nodes (otherwise we can combine them). The size of the CHM M is the size of its DAG representation and all the machines in it. Two important parameters for a CHM is the *depth* m of the DAG (length of the longest path) and the *width* w which is the maximum number of components of a concurrent node of the DAG (i.e. node labelled $\parallel$).

3.1 Expressiveness

Concurrent hierarchical state machines still define of course only regular languages. If a CHM H has width w and depth m, and every FSM in its definition

has n states, then $flat(H)$ has $O(n^{w^m})$ states, that is, flattening causes now in general a double exponential blow up.

This is in general unavoidable. In fact concurrency adds two exponentials even compared to hierarchical state machines: There is a family of languages that can be recognized by linear size CHM's but which require HSM's (and hence also basic FSM's) of double exponential size. Furthermore, there is a triple exponential gap in the worst case between CHM's and deterministic hierarchical state machines (and hence also deterministic FSM's). An example of such a language family is $L = \{w_0\#w_1\#\cdots\#w_k||w_i| = 2^n$ for each i and $w_i = w_j$ for some i, j $\}$.

The following figure summarizes the expressibility relationships between the different kinds of machines.

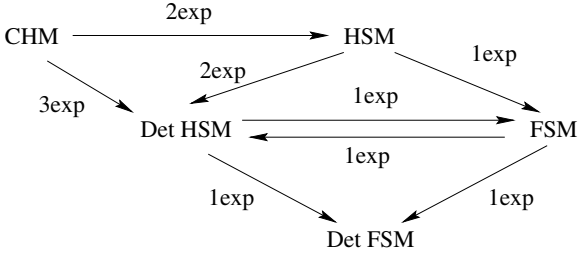


Fig. 3. Summary of succinctness relations

3.2 Operations

Emptiness (Reachability). This cannot be done efficiently any more. Even for a simple concurrent combination (intersection) of basic deterministic FSM's we know that the problem is PSPACE-complete [Koz77], and we saw in the previous section that the same is true for the concurrent combination of just two HSM's. Mixing concurrency with hierarchy adds another exponential: The reachability (and emptiness) problem for CHM's is EXPSPACE-complete.

There are two exponential contributions to this complexity, one due to concurrency and one due to hierarchy. For certain classes of machines, we can avoid one of the exponentials, reducing the complexity to 'only' PSPACE. For example, one case is when all the concurrency is at the top. That is, we have a set of concurrent interacting components, where each component is represented by a hierarchical state machine. Reachability is in this case in PSPACE (and is of course PSPACE-complete). Another case is when the hierarchical construction at a level does not expose the internals of the submachines to higher levels. Note that in the concurrent composition the alphabets of the components are critical. Informally, call a CHM H "well-structured" if the hierarchical nesting always hides the alphabet of the nested machines as far as further concurrent

	Emptiness	Intersection	Universality	Inclusion	Equivalence
FSM	NL	NL	PSPACE	PSPACE	PSPACE
Det FSM	NL	NL	NL	NL	NL
HSM	P	PSPACE	EXPSPACE	EXPSPACE	EXPSPACE
Det HSM	P	PSPACE	P	PSPACE	$\in$ PSPACE
CHM	EXPSPACE	EXPSPACE	2EXPSPACE	2EXPSPACE	2EXPSPACE

Fig. 4. Summary of complexity of operations

combinations with other machines are concerned, that is, in every concurrent combination $M_1 \parallel \cdots \parallel M_t$ in the definition of H (where M_i are without loss of generality hierachical nodes), only the alphabets of the top level components M_i enter in the composition; i.e., the alphabet of the nested submachines are hidden. Then reachability can be solved in PSPACE and in time $O(kn^w)$, where k is the number of operators (internal nodes of the DAG), w is the width, and n is the maximum number of nodes of a FSM in the definition of H.

Universality. The universality problem turns out to be again harder, and is as bad as it can get: It is complete for double exponential space.

Intersection. The intersection of two CHM's H_1, H_2 is itself another CHM $H_1 \parallel H_2$. Emptiness can be done in EXPSPACE (and is complete).

Equivalence, Inclusion. Since universality is hard, the same is true of equivalence and inclusion, which are also complete for double exponential space.

Finally, model checking for a CHM and a Buchi automaton is, as in the case of simple reachability, complete for exponential space (with respect to the size of the model), i.e., if we nest arbitrarily concurrency and hierarchy we have to pay in general for both the concurrency and the hierarchy. It is worth noting however, that these lower bound constructions that give this pessimistic complexity, take advantage of the fact that one can define succinctly (because of the reuse) systems with exponentially many parallel components; this may be unrealistic for significant sizes, and hence the lower bounds may be unduly pessimistic.

The table of Figure 4 summarizes the complexity results of basic operations for the different kinds of machines.

4 Hierarchical Message Sequence Charts

These represent the extreme case where all the concurrency is at the bottom of the hierarchy. A *basic message sequence chart* (MSC) represents the partial order of the message exchanges in a concurrent execution of a set of processes, see Figure 5. The vertical lines represent the processes, time proceeds down vertically for each process, and the arrows represent the messages. Send and receive events are labelled from some alphabet Σ (for example, the messages sent or received, or any other labels). An MSC M represents a set of linear executions, namely all the linearizations of the partial order, and thus it has a corresponding finite language $L(M)$, the strings that label these linearizations.

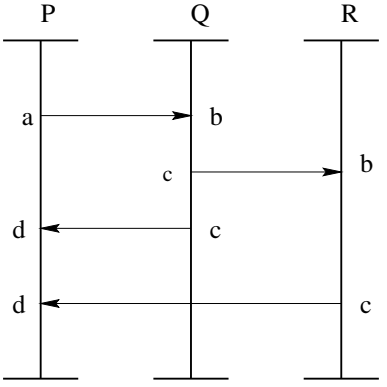


Fig. 5. A basic message sequence chart

Using simple MSC's as basic building blocks, one can combine them in a hierarchical fashion, the same way as with hierarchical state machines. We give for simplicity the definition for the single entry - single exit case (which can be extended to the multiple entries and exits). An *MSC-graph* is a directed graph, with a distinguished initial (entry) node and a final (exit) node, along with a mapping μ that associates with every node a basic MSC (labelled from an alphabet Σ). More generally, we can define inductively a hierarchical MSC graph (HMSC) to be a hierarchical combination of previously defined HMSC's. An HMSC can be represented by a rooted DAG, where each leaf is labelled by a basic MSC's, and each internal node u is labelled by a graph G_u and a mapping μ_u from the nodes of G_u to the children of u. A hierachical MSC H can be of course flattened to a simple MSC-graph $flat(H)$ (at an exponential blow up), the same way that a hierarchical machine is flattened to an FSM.

The executions of a system modeled by an MSC-graph (or HMSC) correspond to the paths of the graph starting at the initial node. There are two interpretations for the concatenation of the MSC's of the different nodes along the path. In the synchronous interpretation, the concatenation of two nodes u, v is viewed as consisting of executing all actions of the MSC of u followed by the actions of the MSC of v; that is, when executing a path, every node represents a block of activity that completes before going on to the next block. In the asynchronous interpretation, the MSC's of the nodes along the path are pasted together, separately process by process, i.e. by identifying for each process the end of its line in one node with the beginning of its line in the next node, thus forming a long MSC spanning all the actions of all the nodes. The synchronous concatenation is probably closer to the way system engineers and designers think when they draw HMSC's partitioning the activities into blocks. The asynchronous interpretation is however closer to what one would get if one was to implement directly the HMSC, i.e. without any other coordination or constraints than what is depicted explicitly in the HMSC.

For each of the interpretations, we can define the language of an HMSC, by considering all the paths starting at the initial node and taking the labelings of all the corresponding linear executions. (We can either define a language of finite strings, if we require that the paths terminate at a final node, or define a language of potentially infinite strings).

In the synchronous interpretation, the language of an HMSC H is regular. A hierarchical state machine H' can be constructed from H by replacing in the representation of the HMSC each basic MSC (ie. leaf of the DAG) by a FSM that recognizes the language of the MSC. Then $L(H) = L(H')$. This is a very benign and convenient case, because we can use directly the results for hierarchical state machines; it means in particular that the graphs and the hierarchy of the HMSC's do not add anything to the complexity of problems like reachability, model checking etc. beyond what's in the simple basic MSC's. If an individual basic MSC is a 'large' one, then the translation to an automaton, as well as answering basic questions about the MSC can be expensive because of the concurrency. Namely, if an MSC M has k processes with l events in each process, then the equivalent FSM can have l^k states, and in fact answering a simple question such as, 'does a given string x belong to $L(M)$' is NP-complete. However, in many cases in hierarchical MSC's most of the size of the HMSC is in the use case graph; the individual MSC's are not large and the number of processes is often small (for example in telephony, processes correspond to entities such as originating and terminating switches and users).

The hierarchy does not introduce any additional complications beyond the basic MSC's under the synchronous interpretation. Thus, for example reachability, or membership, or the model checking problem for a hierarchical HMSC H of size n and a Buchi automaton A of size a is still "only" NP-complete, and can be solved in time $O(a^2 n l^k)$, where k and l are as above the number of processes and number of events per process in a basic MSC. Thus, hierarchical MSC's can be analyzed under the synchronous interpretation without paying a penalty for the hierarchy, and if the basic MSC's are small or do not have too much concurrency (which is often the case) then the complexity is low. Furthermore, for properties that are linearization-independent (i.e., if the property is such that one linearization of an ordinary MSC satisfies the property iff any other one does), then the HMSC H can be checked in linear time in its size: substitute each basic MSC in the definition of H by an automaton (just a path) that accepts just one linearization (instead of all of them); the resulting hierarchical machine satisfies the property iff the given hierarchical MSC does (for linearization-independent properties, there is no difference between synchronous and asynchronous interpretation as far as satisfaction of the properties is concerned, so this holds also for the asynchronous case).

In the asynchronous interpretation, the language of an HMSC is in general not regular, and in fact model checking an MSC-graph for a property given by a Buchi automaton is undecidable. There is a syntactic condition that ensures that the language is regular, and which leads to decidability. Given a set S of basic MSC's, we can construct a corresponding *communication graph* $CG(S)$ among

the processes: this is a directed graph with the processes as nodes and with an arc $P_i \to P_j$ if process P_i sends a message to process P_j in some MSC of S. Given an MSC-graph G (respectively, a hierarchical HMCS H) we say that it is *bounded* if for every cycle K of G (resp. of $flat(H)$) that is reachable from the initial state, the corresponding communication graph $CG(K)$ consists of one strongly connected component and possibly some isolated nodes. If an HMCS is bounded then its language is regular [AY99]. Conversely, it is shown in [HMNT00] that every regular MSC language that is definable by an MSG graph (or equivalently, is finitely generated) then it can be defined by a bounded MSC graph. Boundedness of an HMSC can be determined without flattening the hierarchy in time $O(nl2^k)$ where n is the size of the HMSC, k is the number of processes and l is the number of events per process in a basic MSC (this mild exponential dependence on k is unavoidable, as the boundedness problem is coNP-complete). We can analyze algorithmically bounded HMSC's by constructing an equivalent FSM. The complexity is higher in this case: model checking a Buchi automaton property is PSPACE-complete for bounded MSC graphs and EXPSPACE-complete for HMSC's (the problem is hard even for a fixed property, a fixed number of processes k, and fixed number of events l per basic MSC).

5 Testing

In testing, we have a design model M (eg., a finite state machine) and the actual system S (the implementation under test) and we wish to generate a suitable set of test cases based on the model, which will be applied to the system S to test its correctness, i.e., whether it conforms with the model. There is extensive theoretical and practical work on test generation for FSM's, see for instance [LY96] for a survey. There is a variety of testing criteria that can be used depending on the extent of testing that can be afforded; it can range from checking sequences that provide certain guarrantees on the relation between M and S (usually at a rather high cost), to a more commonly applied simpler set of coverage criteria, such as covering all the states and transitions of the model.

Suppose that we have a hierarchical finite state machine model and let's consider the most common criterion of 'transition coverage', i.e. finding a set of test sequences (paths from the initial state) that cover all the transitions. There are two interpretations of this requirement: in the first interpretation we seek to cover all transitions of M and its nested submachines in all possible contexts, i.e. if a submachine is reused several times, we want to cover all its transitions in all the uses; this is equivalent to covering all the transitions of the flattened machine $flat(M)$. A covering test set can be computed in time polynomial in the size of the flat FSM using flow techniques to minimize the number of test sequences and the total length of the tests. Such an algorithm has been implemented in Lucent's uBET toolset; after flattening the hierarchical MSC graph, an optimal test set is generated, and for each test path, the basic MSC's of the nodes along the path are combined to form a test scenario in the form of an MSC.

An alternative (and more economical) interpretation of the transition coverage requirement is to cover each transition of the hierarchical state machine and each nested submachine at least once overall, i.e., if a submachine is used several times, we only need to cover each transition in only one occurrence (and different transitions could be covered in different copies). One justification for this could be for example if the same implementation of the submachine is used in the different copies, in which case testing a transition in one instance suggests that it will likely work properly in the other instances as well (although that is not necessarily guarranteed). Of course, the number of tests needed under this weaker requirement is generally much smaller than the full coverage requirement; for example, the number of test cases needed is certainly bounded by the size of the hierarchical machine, as opposed to the potentially exponentially larger size of the flattened machine. Computing an optimal (minimum cardinality) test set for the weaker coverage is much harder than for simple FSM's. For example, even in the 2-level case the problem is at least as hard as Set Cover (hence it is hard to compute or to approximate the smallest test set within a constant factor).

Generally, very little systematic work has been done on test generation for hierarchical state machines.

6 Conclusions

Hierarchy is a useful construct in enhancing the expressive power of finite state machines and facilitating the modeling of large systems. We summarized recent work on the theory of hierarchical finite state machines and related specification mechanisms (HMSC's). We discussed the effect of concurrency and nondeterminism, the classification of their expressive power, and the complexity of various algorithmic problems. The picture that emerged is rather comprehensive. One specific problem that remains open is the equivalence of deterministic HSM's. More broadly, we did not touch on the interaction with variables (extended HSM's). Also, as we mentioned more work remains to be done on the testing of hierarchical FSM's.

Another issue concerns design problems, such as the structuring and modularization of the behavioral aspects of systems, to form a suitable hierarchical model. For example, suppose that we have a legacy system which we want to capture formally in a requirement model like uBET. We have available or can generate a large number of executions of the system to obtain a library of MSC scenarios. How do we go from this collection of MSC's to a more higher level, structured view of the system, in the form of a hierarchical use case graph built from basic MSC's?

Similarly, suppose that we have a test model of a system in the form of a large FSM; for example, we produced recently automatically such a model for a large Lucent enterprise switch from the test platform in use for the testing of the switch [EY00]; the FSM model has hundreds of thousands of states. The model can be used for targeted test generation to meet a variety of criteria. However, the

machine is obviously too large for a person to look at, or to manipulate directly, for example to modify as the system evolves, to add new features etc. Can we structure such an FSM and obtain an equivalent (hopefully much smaller) hierarchical FSM? There is an elegant classical decomposition theory of FSM's [HS66] which may be useful in this context.

References

[AHP96] R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.

[AKY99] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. *Proc. 26th Intl. Coll. on Automata, Languages and Programming*, 1999.

[AY98] R. Alur, and M. Yannakakis. Model checking of hierarchical state machines. In *Proc. Sixth Symp. on Foundations of Software Engineering*, pp. 175–188. 1998.

[AY99] R. Alur, and M. Yannakakis. Model checking of message sequence charts. In *Proc. 10th CONCUR*, Springer Verlag, 1999.

[Apf95] L. Apfelbaum. Automated functional test generation. In *Proc. IEEE Autotestcon Conference*, 1995. See also, www.teradyne.com/prods/sst/product_center/t_main.html.

[BVW94] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *Computer Aided Verification, Proc. 6th Int. Conference*, LNCS 818, pages 142–155, 1994.

[BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. Unified Modeling Language User Guide. Addison Wesley, 1997.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. *Proc. CONCUR'97*, LNCS 1243, 1997.

[BS92] O. Burkart and B. Steffen. Model checking for context-free processes. *Proc. CONCUR'92*, LNCS 630, pp. 123-137, 1992.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.

[CVWY92] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory efficient algorithms for the verification of temporal properties. *Formal Methods in System Design*, 1:275–288, 1992.

[DH94] D. Drusinsky and D. Harel. On the power of bounded concurrency I: finite automata. JACM 41(3), 1994.

[EY00] K. Etessami, and M. Yannakakis. From rule-based to automata-based testing. submitted, 2000.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HMNT00] J. G. Henriksen, M. Mukund, K. Narayan Kumar, P. S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. *27th Intl. Coll. on Automata, Languages and Programming*, 2000.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[HPR97] G.J. Holzmann, D. A. Peled, M. H. Redberg. Design tools for requirements engineering. *Bell Labs Technical Journal*, 2(1):86–95, 1997.

[HS66] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, 1966.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley, 1979.

[Jac92] I. Jacobson. *Object-oriented software engineering: a use case driven approach*. Addison-Wesley, 1992.

[JM87] F. Jahanian and A.K. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Trans. on Computers*, C-36(8):961–975, 1987.

[Koz77] D. Kozen. Lower bounds for natural proof systems. *Proc. 18th Annual IEEE Symp. on Foundations of Computer Science*, pp. 254-266, 1977.

[Kur94] R.P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.

[LHHR94] N.G. Leveson, M. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process control systems. *IEEE Trans. on Software Engineering*, 20(9), 1994.

[LY96] D. Lee and M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. *Proceedings of the IEEE*, 84 (8), pp. 1090-1126, 1996.

[McM93] K. McMillan. *Symbolic model checking: An approach to the sate explosion problem.* Kluwer Academic, 1993.

[MSC96] ITU-T Recommendation Z.120. *Message Sequence Chart (MSC)*. 1996.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the 18th IEEE Symposium on Foundations of Computer Science*, pages 46–77, 1977.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. *Proc. of the 5th Intl. Symp. on Programming*, LNCS 137, pp. 195–220, 1982.

[RBPE91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. *Object-oriented modeling and design*. Prentice-Hall, 1991.

[RGG96] E. Rudolph, J. Grabowski, P. Graubmann. Tutorial on Message Sequence Charts (MSC'96). *FORTE/PSTV'96*, 1996.

[SGW94] B. Selic, G. Gullekson, and P. T. Ward. *Real-time object oriented modeling and design*. J. Wiley, 1994.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. *Proc. of the First IEEE Symp. on Logic in Computer Science*, pp. 332–344, 1986.

Ambient Groups and Mobility Types

Luca Cardelli[1], Giorgio Ghelli[2], and Andrew D. Gordon[1]

[1] Microsoft Research
[2] Pisa University

Abstract. We add name groups and group creation to the typed ambient calculus. Group creation is surprisingly interesting: it has the effect of statically preventing certain communications, and can thus block the accidental or malicious escape of capabilities that is a major concern in practical systems. Moreover, ambient groups allow us to refine our earlier work on type systems for ambient mobility. We present type systems in which groups identify the set of ambients that a process may cross or open.

1 Introduction

The Ambient Calculus is a process calculus based on local communication and on process mobility. The basic, untyped, calculus can be decorated with static information to restrict either local communication, or mobility, or both.

Exchange control systems can be used to restrict communication. In [CG99] we have investigated *exchange types*, which subsume standard type systems for processes and functions, but do not impose restrictions on mobility.

Mobility control systems can be used to restrict mobility. In [CGG99] we investigate *immobility* and *locking* annotations, which are simple predicates about mobility.

The goal of this paper is to refine our previous work on mobility control, by including in the type of a process static descriptions of the set of ambients it may cross, and the set of ambients it may open. To do so, we adopt a new construction of independent interest. Among the types, we introduce collections of names that we call *groups*; names belong to groups in the same sense that values belong to types.

To understand how name groups arise, consider a typical static property we may want to express in a type system for the ambient calculus, informally:

> The ambient named n can enter the ambient named m.

This could be expressed as a typing $n : \mathit{CanEnter}(m)$ stating that n is a member of the collection $\mathit{CanEnter}(m)$ of names that can enter m. However, this would bring us straight into the domain of dependent types, since the type $\mathit{CanEnter}(m)$ depends on the name m. Instead, we introduce type-level groups of names, G, H, and restate our property as:

> The name n belongs to group G; the name m belongs to group H. Any ambient of group G can enter any ambient of group H.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 333–347, 2000.
© Springer-Verlag Berlin Heidelberg 2000

This idea leads to typing judgments of the form:

process P may cross ambients of group G
process P may open ambients of group G

The former reduces to immobility assertions when a process can cross no groups; the latter reduces to locking assertions, when members of a group can be opened by no process [CGG99].

Among the processes, we then introduce an operation, $(\nu G)P$, for creating new groups. Within P we can introduce new names of group G. The binders for new groups, (νG), extrude in much the same way as binders for new names, $(\nu n{:}G)$. Because of extrusion, group binders do not impede the mobility of ambients that are enclosed in the initial scope of fresh groups. However, simple scoping restrictions prevent names of a fresh group from ever being received outside the initial scope of the group.

Therefore, we obtain a flexible way of protecting the propagation of names. This is to be contrasted with the situation in the untyped π-calculus and ambient calculus, where names can (intentionally, accidentally, or maliciously) be extruded arbitrarily far, by the automatic and unrestricted application of extrusion rules.

We organise the paper as follows. In the remainder of this opening section we review the basic untyped ambient calculus. Section 2 describes the typed ambient calculus with groups—obtained by enriching our exchange type system [CG99] with groups. Section 3 enriches the system of Section 2 to control ambient opening. In Section 4, we define a system in which the type of a process records both the groups it may open and the groups it may cross. Section 5 formalizes safety properties guaranteed by typing. Section 6 concludes and discusses related work.

A technical report contains proofs omitted from this paper [CGG00].

1.1 The Untyped Ambient Calculus (Review)

An ambient is a named boundary whose interior contains a collection of running processes, possibly including nested subambients. We explain the untyped ambient calculus elsewhere [CG98] in detail, but here we introduce its central features via a standard example: $a[p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]] \mid b[\mathit{open}\ p.(x).x[]]$.

Intuitively, this example represents a packet named p being sent from a machine a to a machine b. The example consists of the parallel composition (indicated by the $\mid$ operator) of two ambients, named a and b. The brackets $[\ldots]$ represent ambients' boundaries. The process $p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]$ represents the packet, a subambient of ambient a. The name of the packet ambient is p, and its interior is the process $\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle$. This process consists of three sequential actions: exercise the capability $\mathit{out}\,a$, exercise the capability $\mathit{in}\,b$, and then output the name c. The effect of the two capabilities on the enclosing ambient p is to move p out of a and into b, to reach the state: $a[] \mid b[p[\langle c\rangle] \mid \mathit{open}\ p.(x).x[]]$. The interior of a is now empty. The interior of b consists of two running processes, the subambient $p[\langle c\rangle]$ and the process $\mathit{open}\ p.(x).x[]$. The latter is attempting to exercise the $\mathit{open}\,p$ capability. Previously it was blocked. Now that the p ambient is present,

the effect of *open* p is to dissolve the ambient's boundary. Hence, the interior of b becomes the process $\langle c\rangle \mid (x).x[]$. This is a composition of an output $\langle c\rangle$ with an input $(x).x[]$. The input consumes the output, leaving $c[]$ as the interior of b. Hence, the final state of the whole example is $a[] \mid b[c[]]$.

The $\mathbf{0}$ process represents inactivity; the notation $a[]$ for an empty ambient named a, used above, is actually short for $a[\mathbf{0}]$. There are also replication and restriction constructs. A replication $!P$ behaves the same as an unlimited number of parallel copies of P. A restriction $(\nu n)P$ creates a new name n with scope P.

2 The Typed Ambient Calculus with Groups

We start with the typed ambient calculus of [CG99] and we add a new process construct, $(\nu G)P$, to create a new group G with scope P. Correspondingly we add a new type construct, $G[T]$, for the type of names of group G that name ambients that contain T exchanges.

The construct $G[T]$ is actually a refinement of the construct $Amb[T]$ of [CG99], where Amb can now be seen as the group of all names. It is conceivable to introduce a subtype ordering on groups, with Amb as the maximal element. However, subtyping may help capabilities escape, particularly in the presence of a maximal element; we do not consider these extensions in this paper.

We can now write, for example, the following typed process:

$$(\nu\, Ch)(\nu\, Msg)(\nu c{:}Ch[Msg[Shh]])(\nu m{:}Msg[Shh])c[\langle m\rangle \mid (x{:}Msg[Shh]).x[]]$$

This creates two groups Ch and Msg and two names c and m belonging to those groups. The types ensure that only messages, that is, names of type $Msg[Shh]$, can be exchanged inside an ambient named c, as happens in the rest of the process. (The type Shh prohibits exchanges; names of type $Msg[Shh]$ are in group Msg, and name ambients in which exchanges are prohibited.)

The types of the ambient calculus with groups are the same as in [CG99], except that $G[T]$ replaces $Amb[T]$. We have types W for messages. Messages can be either names of type $G[T]$, or capabilities of type $Cap[T]$. We also have types for processes, T, that classify processes according to the type of message tuples they exchange (if any).

Types:

$W ::=$	message type
$\quad G[T]$	$\quad$ ambient name in group G with T exchanges
$\quad Cap[T]$	$\quad$ capability unleashing T exchanges
$S, T ::=$	exchange type
$\quad Shh$	$\quad$ no exchange
$\quad W_1 \times \cdots \times W_k$	$\quad$ tuple exchange ($\mathbf{1}$ is the null product)

Expressions (messages) and processes are also the same as in [CG99], except that we add processes $(\nu G)P$ and include the objective moves of [CGG99].

The movement primitives of the untyped calculus, illustrated by the process $p[out\ a.in\ b.\langle c\rangle]$ from Section 1.1, are called *subjective moves*; the capabilities *out* a and *in* b move the ambient p from the inside. In the typed calculus, we also take *objective moves* as primitive. In an objective move $go N.M[P]$, the capability N moves an ambient $M[P]$ from the outside by following the path encoded by N, and once there starts the ambient $M[P]$. In the untyped calculus, we can define an objective move $go\ N.M[P]$ to be short for the process $(\nu k)k[N.M[out\ k.P]]$ where k is not free in P. As we found in our previous work [CGG99], a primitive typing rule for objective moves allows more refined typings than are possible with only subjective moves.

Expressions and processes:

$M, N ::=$	expression	$P, Q, R ::=$	process
n	name	$(\nu G)P$	group creation
$in\ M$	can enter M	$(\nu n{:}W)P$	restriction
$out\ M$	can exit M	$\mathbf{0}$	inactivity
$open\ M$	can open M	$P \mid Q$	composition
ϵ	null	$!P$	replication
$M.M'$	path	$M[P]$	ambient
		$M.P$	action
		$(x_1{:}W_1, \ldots, x_k{:}W_k).P$	input
		$\langle M_1, \ldots, M_k\rangle$	output
		$go\ N.M[P]$	objective move

This grammar allows the formation of certain nonsensical processes, where a capability is used in place of a name, as in $(in\ n)[\mathbf{0}]$, or vice versa, as in $(\nu n{:}W)n.\mathbf{0}$. Such garbled processes are not typable in any of our type systems.

In the processes $(\nu G)P$ and $(\nu n{:}W)P$, the group G and the name n, respectively, are bound, with scope P. In the process $(x_1{:}W_1, \ldots, x_k{:}W_k).P$, the names $x_1, \ldots, x_k$ are bound, with scope P. We identify processes up to consistent renaming of bound names and bound groups. We write $fn(P)$ for the set of names free in process P, and we write $fg(P)$, $fg(W)$, and $fg(T)$ for the sets of groups free in process P, message type W, and exchange type T, respectively.

The following tables describe the structural congruence rules and the reduction rules. The bottom four rules of structural congruence describe the extrusion behavior of the (νG) binders. Side conditions on these rules prevent violation of lexical scoping. The notation $P\{x_1{\leftarrow}M_1, \ldots, x_k{\leftarrow}M_k\}$ used below in the reduction rule for I/O denotes the outcome of a capture-avoiding simultaneous substitution, for each $i \in 1..k$, of the expression M_i for each free occurrence of the corresponding name x_i in the process P.

Structural Congruence:

$P \equiv Q \Rightarrow (\nu n{:}W)P \equiv (\nu n{:}W)Q$	$P \equiv P$
$P \equiv Q \Rightarrow (\nu G)P \equiv (\nu G)Q$	$Q \equiv P \Rightarrow P \equiv Q$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$

$P \equiv Q \Rightarrow\ !P \equiv\ !Q$
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$ $\quad P \mid Q \equiv Q \mid P$
$P \equiv Q \Rightarrow M.P \equiv M.Q$ $\quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$P \equiv Q \Rightarrow go\ N.M[P] \equiv go\ N.M[Q]$
$P \equiv Q \Rightarrow (x_1{:}W_1, \ldots, x_k{:}W_k).P \equiv (x_1{:}W_1, \ldots, x_k{:}W_k).Q$

$n_1 \neq n_2 \Rightarrow (\nu n_1{:}W_1)(\nu n_2{:}W_2)P \equiv (\nu n_2{:}W_2)(\nu n_1{:}W_1)P$
$n \notin fn(P) \Rightarrow (\nu n{:}W)(P \mid Q) \equiv P \mid (\nu n{:}W)Q$
$n \neq m \Rightarrow (\nu n{:}W)m[P] \equiv m[(\nu n{:}W)P]$

$P \mid \mathbf{0} \equiv P$ $\quad !P \equiv P \mid !P$
$(\nu n{:}W)\mathbf{0} \equiv \mathbf{0}$ $\quad \epsilon.P \equiv P$
$(\nu G)\mathbf{0} \equiv \mathbf{0}$ $\quad (M.M').P \equiv M.M'.P$
$!\mathbf{0} \equiv \mathbf{0}$ $\quad go\ \epsilon.M[P] \equiv M[P]$

$(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$
$G \notin fg(W) \Rightarrow (\nu G)(\nu n{:}W)P \equiv (\nu n{:}W)(\nu G)P$
$G \notin fg(P) \Rightarrow (\nu G)(P \mid Q) \equiv P \mid (\nu G)Q$
$(\nu G)m[P] \equiv m[(\nu G)P]$

Reduction:

$n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]$ $\quad P \to Q \Rightarrow (\nu G)P \to (\nu G)Q$
$m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]$ $\quad P \to Q \Rightarrow (\nu n{:}W)P \to (\nu n{:}W)Q$
$open\ n.P \mid n[Q] \to P \mid Q$ $\quad P \to Q \Rightarrow P \mid R \to Q \mid R$
$\langle M_1, \ldots, M_k \rangle \mid (x_1{:}W_1, \ldots, x_k{:}W_k).P$ $\quad P \to Q \Rightarrow n[P] \to n[Q]$
$\quad \to P\{x_1 \leftarrow M_1, \ldots, x_k \leftarrow M_k\}$ $\quad P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$
$go\ (in\ m.N).n[P] \mid m[Q] \to m[go\ N.n[P] \mid Q]$
$m[go\ (out\ m.N).n[P] \mid Q] \to go\ N.n[P] \mid m[Q]$

Next, we introduce the five basic judgments and the typing rules. Apart from minor adaptations, the main novelty with respect to [CG99] is the rule with conclusion $E \vdash (\nu G)P : T$. The assumptions of this rule are that $E, G \vdash P : T$ and $G \notin fg(T)$. The latter assumption prevents G from going out of scope in the conclusion. Typing environments, E, are given by the grammar $E ::= \varnothing \mid E, G \mid E, n{:}W$. For each E, we inductively define $dom(E)$ by the equations $dom(\varnothing) = \varnothing$, $dom(E, G) = dom(E) \cup \{G\}$, and $dom(E, n{:}W) = dom(E) \cup \{n\}$.

Judgments:

$E \vdash \diamond$	good environment
$E \vdash W$	good message type W
$E \vdash T$	good exchange type T
$E \vdash M : W$	good expression M of message type W
$E \vdash P : T$	good process P with exchange type T

Typing Rules:

$$\frac{}{\varnothing \vdash \diamond} \qquad \frac{E \vdash W \quad n \notin dom(E)}{E, n{:}W \vdash \diamond} \qquad \frac{E \vdash \diamond \quad G \notin dom(E)}{E, G \vdash \diamond}$$

$$\frac{G \in dom(E) \quad E \vdash T}{E \vdash G[T]} \qquad \frac{E \vdash T}{E \vdash Cap[T]} \qquad \frac{E \vdash \diamond}{E \vdash Shh} \qquad \frac{E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash W_1 \times \cdots \times W_k}$$

$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W} \qquad \frac{E \vdash Cap[T]}{E \vdash \epsilon : Cap[T]} \qquad \frac{E \vdash M : Cap[T] \quad E \vdash M' : Cap[T]}{E \vdash M.M' : Cap[T]}$$

$$\frac{E \vdash n : G[S] \quad E \vdash T}{E \vdash in\ n : Cap[T]} \qquad \frac{E \vdash n : G[S] \quad E \vdash T}{E \vdash out\ n : Cap[T]} \qquad \frac{E \vdash n : G[T]}{E \vdash open\ n : Cap[T]}$$

$$\frac{E \vdash M : Cap[T] \quad E \vdash P : T}{E \vdash M.P : T} \qquad \frac{E \vdash M : G[S] \quad E \vdash P : S \quad E \vdash T}{E \vdash M[P] : T}$$

$$\frac{E, n{:}G[S] \vdash P : T}{E \vdash (\nu n{:}G[S])P : T} \qquad \frac{E, G \vdash P : T \quad G \notin fg(T)}{E \vdash (\nu G)P : T} \qquad \frac{E \vdash T}{E \vdash \mathbf{0} : T} \qquad \frac{E \vdash P : T}{E \vdash !P : T}$$

$$\frac{E \vdash P : T \quad E \vdash Q : T}{E \vdash P \mid Q : T} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k}{E \vdash \langle M_1, \ldots, M_k \rangle : W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[S'] \quad E \vdash M : G[S] \quad E \vdash P : S \quad E \vdash T}{E \vdash go\ N.M[P] : T}$$

We obtain a standard subject reduction result. A subtle point, though, is the need to account for the appearance of new groups (G_1, ..., G_k, below) during reduction. This is because reduction is defined up to structural congruence, and structural congruence does not preserve the set of free groups of a process. The culprit is the rule $(\nu n{:}W)\mathbf{0} \equiv \mathbf{0}$, in which groups free in W are not free in $\mathbf{0}$.

Theorem 1. *If $E \vdash P : T$ and either $P \equiv Q$ or $P \to Q$ then there are G_1, ..., G_k such that $G_1, \ldots, G_k, E \vdash Q : T$.*

3 Opening Control

In this section, to control usage of the *open* capability, we add attributes to the ambient types, $G[T]$, and the capability types, $Cap[T]$, of the previous type system. (In the next section, to control usage of the *in* and *out* capabilities, we add further attributes.)

To control the opening of ambients, we formalize the constraint that the name of any ambient opened by a process is in one of the groups G_1, ..., G_k,

but in no others. To do so, we add an attribute $^\circ\{G_1, \ldots, G_k\}$ to ambient types, which now take the form $G[^\circ\{G_1, \ldots, G_k\}, T]$. A name of this type is in group G, and names ambients within which processes may exchange messages of type T and may only open ambients in the groups $G_1, \ldots, G_k$. We need to add the same attribute to capability types, which now take the form $Cap[^\circ\{G_1, \ldots, G_k\}, T]$. Exercising a capability of this type may unleash exchanges of type T and openings of ambients in groups $G_1, \ldots, G_k$. The typing judgment for processes acquires the form $E \vdash P : ^\circ\{G_1, \ldots, G_k\}, T$. The pair $^\circ\{G_1, \ldots, G_k\}, T$ constrains both the *opening effects* (what ambients the process opens) and the *exchange effects* (what messages the process exchanges). We call such a pair an *effect*, and introduce the metavariable F to range over effects. It is also convenient to introduce metavariables $\mathbf{G}$, $\mathbf{H}$ to range over finite sets of name groups. The following table summarizes these metavariable conventions and our enhanced syntax for types:

Group Sets and Types:

$\mathbf{G}, \mathbf{H} ::= \{G_1, \ldots, G_k\}$	finite set of name groups
$W ::=$	message type
$\quad G[F]$	$\quad$ ambient name in group G (contains processes with F effects)
$\quad Cap[F]$	$\quad$ capability (unleashes F effects)
$F ::=$	effect
$\quad ^\circ\mathbf{H}, T$	$\quad$ may open $\mathbf{H}$, may exchange T
$S, T ::=$	exchange type
$\quad Shh$	$\quad$ no exchange
$\quad W_1 \times \cdots \times W_k$	$\quad$ tuple exchange

The following tables define the type system in detail. There are five basic judgments as before. They have the same format except that the judgment $E \vdash F$, meaning that the effect F is good given environment E, replaces the previous judgment $E \vdash T$. We omit the three rules for deriving good environments; they are exactly as in the previous section. There are two main differences between the other rules below and the rules of the previous section. First, effects, F, replace exchange types, T, throughout. Second, in the rule ascribing a type to *open* n, the condition $G \in \mathbf{H}$ constrains the opening effect $\mathbf{H}$ of the capability *open* n to include the group G, the group of the name n.

Judgments:

$E \vdash \diamond$	good environment
$E \vdash W$	good message type W
$E \vdash F$	good effect F
$E \vdash M : W$	good expression M of message type W
$E \vdash P : F$	good process P with F effects

Typing Rules:

$$\frac{G \in dom(E) \quad E \vdash F}{E \vdash G[F]} \qquad \frac{E \vdash F}{E \vdash Cap[F]} \qquad \frac{\mathbf{H} \subseteq dom(E) \quad E \vdash \diamond}{E \vdash {}^{\circ}\mathbf{H}, Shh}$$

$$\frac{\mathbf{H} \subseteq dom(E) \quad E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k} \qquad \frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W}$$

$$\frac{E \vdash Cap[F]}{E \vdash \epsilon : Cap[F]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash M' : Cap[F]}{E \vdash M.M' : Cap[F]}$$

$$\frac{E \vdash n : G[F] \quad E \vdash {}^{\circ}\mathbf{H}, T}{E \vdash in\ n : Cap[{}^{\circ}\mathbf{H}, T]} \qquad \frac{E \vdash n : G[F] \quad E \vdash {}^{\circ}\mathbf{H}, T}{E \vdash out\ n : Cap[{}^{\circ}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G[{}^{\circ}\mathbf{H}, T] \quad G \in \mathbf{H}}{E \vdash open\ n : Cap[{}^{\circ}\mathbf{H}, T]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash P : F}{E \vdash M.P : F}$$

$$\frac{E \vdash M : G[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash M[P] : F'} \qquad \frac{E, n{:}G[F] \vdash P : F'}{E \vdash (\nu n{:}G[F])P : F'}$$

$$\frac{E, G \vdash P : F \quad G \notin fg(F)}{E \vdash (\nu G)P : F} \qquad \frac{E \vdash F}{E \vdash \mathbf{0} : F} \qquad \frac{E \vdash P : F}{E \vdash !P : F}$$

$$\frac{E \vdash P : F \quad E \vdash Q : F}{E \vdash P \mid Q : F} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k \quad \mathbf{H} \subseteq dom(E)}{E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[{}^{\circ}\mathbf{H}, T] \quad E \vdash M : G[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash go\ N.M[P] : F'}$$

Theorem 2. *If $E \vdash P : F$ and either $P \equiv Q$ or $P \to Q$ then there are $G_1, \ldots, G_k$ such that $G_1, \ldots, G_k, E \vdash Q : F$.*

Here is a simple example of a typing derivable in this system:

$$G, n{:}G[{}^{\circ}\{G\}, Shh] \vdash n[\mathbf{0}] \mid open\ n.\mathbf{0} : {}^{\circ}\{G\}, Shh$$

This asserts that the whole process $n[\mathbf{0}] \mid open\ n.\mathbf{0}$ is well-typed and opens only ambients in the group G.

On the other hand, one might expect the following variant to be derivable, but it is not:

$$G, n{:}G[{}^{\circ}\varnothing, Shh] \not\vdash n[\mathbf{0}] \mid open\ n.\mathbf{0} : {}^{\circ}\{G\}, Shh$$

This is because the typing rule for *open n* requires the effect unleashed by the *open n* capability to be the same as the effect contained within the ambient n.

But the opening effect $^\circ\varnothing$ specified by the type $G[^\circ\varnothing, Shh]$ of n cannot be the same as the effect unleashed by *open* n, because the rule also requires the latter to at least include the group G of n.

We have not found this feature to be problematic, and indeed it has a positive side-effect: the type $G[^\circ\mathbf{G}, T]$ of an ambient name n not only tells which opening effects may happen inside the ambient, but also tells whether n may be opened from outside: it is openable only if $G \in \mathbf{G}$, since this is the only case when *open* $n.\mathbf{0} \mid n[P]$ can be well typed. Hence, the presence of G in the set $\mathbf{G}$ may either mean that n is meant to be an ambient within which other ambients in group G may be opened, or that it is meant to be an openable ambient.

4 Crossing Control

This section presents the third and final type system of the paper, obtained by enriching the type system of Section 3 with attributes to control mobility.

Movement operators enable an ambient n to cross the boundary of another ambient m either by entering it via an *in* m capability or by exiting it via an *out* m capability. In the type system of this section, the type of n lists those groups that may be crossed; the ambient n may only cross the boundary of another ambient m if the group of m is included in this list. In our typed calculus, there are two kinds of movement, subjective moves and objective moves. Therefore, we separately list those groups that may be crossed by objective moves and those groups that may be crossed by subjective moves.

We add new attributes to the syntax of ambient types, effects, and capability types. An ambient type acquires the form $G\,{}^\frown\mathbf{G}'[{}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T]$. An ambient of this type is in group G, may cross ambients in groups $\mathbf{G}'$ by objective moves, may cross ambients in groups $\mathbf{G}$ by subjective moves, may open ambients in groups $\mathbf{H}$, and may contain exchanges of type T. An effect, F, of a process is now of the form ${}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T$. It asserts that the process may exercise *in* and *out* capabilities to accomplish subjective moves across ambients in groups $\mathbf{G}$, that the process may open ambients in groups $\mathbf{H}$, and that the process may exchange messages of type T. Finally, a capability type retains the form $Cap[F]$, but with the new interpretation of F. Exercising a capability of this type may unleash F effects.

Types:

$W ::=$	message type
$\quad G\,{}^\frown\mathbf{G}[F]$	ambient name in group G, crosses $\mathbf{G}$ objectively, contains processes with F effects
$\quad Cap[F]$	capability (unleashes F effects)
$F ::=$	effect
$\quad {}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T$	crosses $\mathbf{G}$, opens $\mathbf{H}$, exchanges T
$S, T ::=$	exchange type
$\quad Shh$	no exchange
$\quad W_1 \times \cdots \times W_k$	tuple exchange

The format of the five judgments making up the system is the same as in Section 3. We omit the three rules defining good environments; they are as in Section 2. There are two main changes to the previous system to control mobility. First, the rules for typing *in* n and *out* n change to assign a type $Cap[\frown\mathbf{G},\mathring{}\mathbf{H},T]$ to the capabilities *in* n and *out* n only if $G \in \mathbf{G}$ where G is the group of n. Second, the rule for objective moves changes to allow an objective move of an ambient of type $G\frown\mathbf{G}'[F]$ by a capability of type $Cap[\frown\mathbf{G},\mathring{}\mathbf{H},T]$ only if $\mathbf{G} = \mathbf{G}'$.

Typing Rules:

$$\frac{G \in dom(E) \quad \mathbf{G} \subseteq dom(E) \quad E \vdash F}{E \vdash G\frown\mathbf{G}[F]} \qquad \frac{E \vdash F}{E \vdash Cap[F]}$$

$$\frac{\mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E) \quad E \vdash \diamond}{E \vdash \frown\mathbf{G},\mathring{}\mathbf{H}, Shh}$$

$$\frac{\mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E) \quad E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash \frown\mathbf{G},\mathring{}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W} \qquad \frac{E \vdash Cap[F]}{E \vdash \epsilon : Cap[F]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash M' : Cap[F]}{E \vdash M.M' : Cap[F]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[F] \quad E \vdash \frown\mathbf{G},\mathring{}\mathbf{H}, T \quad G \in \mathbf{G}}{E \vdash in\ n : Cap[\frown\mathbf{G},\mathring{}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[F] \quad E \vdash \frown\mathbf{G},\mathring{}\mathbf{H}, T \quad G \in \mathbf{G}}{E \vdash out\ n : Cap[\frown\mathbf{G},\mathring{}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[\frown\mathbf{G},\mathring{}\mathbf{H}, T] \quad G \in \mathbf{H}}{E \vdash open\ n : Cap[\frown\mathbf{G},\mathring{}\mathbf{H}, T]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash P : F}{E \vdash M.P : F}$$

$$\frac{E \vdash M : G\frown\mathbf{G}[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash M[P] : F'} \qquad \frac{E, n{:}G\frown\mathbf{G}[F] \vdash P : F'}{E \vdash (\nu n{:}G\frown\mathbf{G}[F])P : F'}$$

$$\frac{E, G \vdash P : F \quad G \notin fg(F)}{E \vdash (\nu G)P : F} \qquad \frac{E \vdash F}{E \vdash \mathbf{0} : F} \qquad \frac{E \vdash P : F}{E \vdash !P : F}$$

$$\frac{E \vdash P : F \quad E \vdash Q : F}{E \vdash P \mid Q : F} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : \frown\mathbf{G},\mathring{}\mathbf{H}, W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : \frown\mathbf{G},\mathring{}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k \quad \mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E)}{E \vdash \langle M_1, \ldots, M_k \rangle : \frown\mathbf{G},\mathring{}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[\frown\mathbf{G},\mathring{}\mathbf{H}, T] \quad E \vdash M : G\frown\mathbf{G}[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash go\ N.M[P] : F'}$$

Theorem 3. *If $E \vdash P : F$ and either $P \equiv Q$ or $P \to Q$ then there are $G_1, \ldots, G_k$ such that $G_1, \ldots, G_k, E \vdash Q : F$.*

Recall the untyped example from Section 1.1. Consider two groups G and H. Let $W = G\,{}^{\frown}\varnothing[{}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh}]$ and set P to be the example process:

$$P = a[p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]] \mid b[\mathit{open}\ p.(x{:}W).x[]]$$

Let $E = G, H, a{:}W, b{:}G\,{}^{\frown}\varnothing[{}^{\frown}\{G\}, {}^{\circ}\{H\}, W], c{:}W, p{:}H\,{}^{\frown}\varnothing[{}^{\frown}\{G\}, {}^{\circ}\{H\}, W]$. Then we can derive the typings:

$$\begin{array}{l} E \vdash \mathit{out}\ a.\mathit{in}\ b.\langle c\rangle : {}^{\frown}\{G\}, {}^{\circ}\{H\}, W \\ E \vdash \mathit{open}\ p.(x{:}W).x[] : {}^{\frown}\{G\}, {}^{\circ}\{H\}, W \\ E \vdash P : {}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh} \end{array}$$

From the typings $a, c : G\,{}^{\frown}\varnothing[{}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh}]$, we can tell that ambients a and c are immobile ambients in which nothing is exchanged and that cannot be opened. From the typings $p{:}H\,{}^{\frown}\varnothing[{}^{\frown}\{G\}, {}^{\circ}\{H\}, W], b{:}G\,{}^{\frown}\varnothing[{}^{\frown}\{G\}, {}^{\circ}\{H\}, W]$, we can tell that the ambients b and p cross only G ambients, open only H ambients, and contain W exchanges; the typing of p also tells us it can be opened. This is good, but is not fully satisfactory, since, if b were meant to be immobile, we would like to express this immobility invariant in its type. However, since b opens a subjectively mobile ambient, then b must be typed as if it were subjectively mobile itself (by the rule for *open*).

As already observed in [CGG99], this problem can be solved by replacing the subjective moves by objective moves. Let $W = G\,{}^{\frown}\varnothing[{}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh}]$, again, and set Q to be the example process with objective instead of subjective moves:

$$Q = a[\mathit{go}\ (\mathit{out}\ a.\mathit{in}\ b).p[\langle c\rangle]] \mid b[\mathit{open}\ p.(x{:}W).x[]]$$

Let $E = G, H, a{:}W, b{:}G\,{}^{\frown}\varnothing[{}^{\frown}\varnothing, {}^{\circ}\{H\}, W], c{:}W, p{:}H\,{}^{\frown}\{G\}[{}^{\frown}\varnothing, {}^{\circ}\{H\}, W]$, and we can derive:

$$\begin{array}{l} E \vdash \mathit{out}\ a.\mathit{in}\ b : \mathit{Cap}[{}^{\frown}\{G\}, {}^{\circ}\varnothing, \mathit{Shh}] \\ E \vdash \mathit{go}\ (\mathit{out}\ a.\mathit{in}\ b).p[\langle c\rangle] : {}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh} \\ E \vdash \mathit{open}\ p.(x{:}W).x[] : {}^{\frown}\varnothing, {}^{\circ}\{H\}, W \\ E \vdash Q : {}^{\frown}\varnothing, {}^{\circ}\varnothing, \mathit{Shh} \end{array}$$

The typings of a and c are unchanged, but the new typings of p and b are more informative. We can tell from the typing $p{:}H\,{}^{\frown}\{G\}[{}^{\frown}\varnothing, {}^{\circ}\{H\}, W]$ that movement of p is now due to objective rather than subjective moves. We can now tell from the typing $b{:}G\,{}^{\frown}\varnothing[{}^{\frown}\varnothing, {}^{\circ}\{H\}, W]$ that the ambient b is immobile.

This example suggests that in some situations objective moves lead to more informative typings than subjective moves. Still, subjective moves are essential for moving ambients containing running processes. We need such ambients to model mobile agents, for example.

5 Upper Bounds on Capabilities Imposed by Effects

Like most other type systems for concurrent calculi, ours does not guarantee liveness, for example, the absence of deadlocks. Still, we may regard the effect assigned to a process as a safety property: an upper bound on the capabilities that may be exercised by the process, and hence on its behavior. We formalize this idea in the setting of our third type system, and explain some consequences.

A similar analysis can be applied to the simpler type system of Section 3.

We say that a process P exercises a capability M, one of *in* n or *out* n or *open* n, just if $P \downarrow M$ may be derived by the following rules:

Exercising a capability: $P \downarrow M$ **where** $M \in \{in\ n, out\ n, open\ n\}$

$$\frac{P \equiv M.Q}{P \downarrow M} \qquad \frac{P \downarrow M}{P \mid Q \downarrow M} \qquad \frac{Q \downarrow M}{P \mid Q \downarrow M} \qquad \frac{P \downarrow M \quad n \notin fn(M)}{(\nu n{:}W)P \downarrow M} \qquad \frac{P \downarrow M}{(\nu G)P \downarrow M}$$

We begin by defining a fragment of a labelled transition system for the ambient calculus [GC99]. We say that a process P exercises a capability M, one of *in* n or *out* n or *open* n, to leave residue P' just if the M-labelled transition $P \xrightarrow{M} P'$ may be derived by the following rules:

Labelled Transitions: $P \xrightarrow{M} P'$ **where** $M \in \{in\ n, out\ n, open\ n\}$

$$\frac{P \equiv M.Q}{P \xrightarrow{M} Q} \qquad \frac{P \xrightarrow{M} P' \quad m \notin fn(M)}{(\nu m{:}W)P \xrightarrow{M} (\nu m{:}W)P'} \qquad \frac{P \xrightarrow{M} P'}{(\nu G)P \xrightarrow{M} (\nu G)P'}$$

$$\frac{P \xrightarrow{M} P'}{P \mid Q \xrightarrow{M} P' \mid Q} \qquad \frac{Q \xrightarrow{M} Q'}{P \mid Q \xrightarrow{M} P \mid Q'}$$

The following asserts that the group of the name contained in any capability exercised by a well-typed process is bounded by the effect assigned to the process. It is a corollary of Theorem 3.

Theorem 4 (Effect Safety). *Suppose that* $E \vdash P : \curvearrowright\mathbf{G}, \circledcirc\mathbf{H}, T$.

(1) *If* $P \downarrow in\ n$ *then* $E \vdash n : G \curvearrowright \mathbf{G}'[F]$ *for some type* $G \curvearrowright \mathbf{G}'[F]$ *with* $G \in \mathbf{G}$.
(2) *If* $P \downarrow out\ n$ *then* $E \vdash n : G \curvearrowright \mathbf{G}'[F]$ *for some type* $G \curvearrowright \mathbf{G}'[F]$ *with* $G \in \mathbf{G}$.
(3) *If* $P \downarrow open\ n$ *then* $E \vdash n : G \curvearrowright \mathbf{G}'[F]$ *for some type* $G \curvearrowright \mathbf{G}'[F]$ *with* $G \in \mathbf{H}$.

To explain the operational significance of this theorem, consider a name $m : H \curvearrowright \mathbf{H}'[\curvearrowright\mathbf{G}, \circledcirc\mathbf{H}, T]$ and a well-typed ambient $m[P]$. Suppose that $m[P]$ is a subprocess of some well-typed process Q. We can show, by adapting standard techniques [GC99], two connections between the M-labelled transitions of the process P and the reductions immediately derivable from the whole process Q.

First, within Q, the ambient $m[P]$ can cross the boundary of another ambient named n of some group G only if either $P \xrightarrow{in\ n} P'$ or $P \xrightarrow{out\ n} P'$ for some P'. The typing rule for ambients implies that P must have effect $\curvearrowright\mathbf{G}, \circledcirc\mathbf{H}, T$. Part (1)

or (2) of the theorem implies that the set $\mathbf{G}$ contains G. Second, suppose that P includes a top-level ambient named n. The boundary of n can be dissolved only if $P \xrightarrow{open\, n} P'$ for some P'. Since P has effect $\curvearrowright\mathbf{G}, \circ\mathbf{H}, T$, part (3) of the theorem implies that the set $\mathbf{H}$ contains G. So the set $\mathbf{G}$ includes the groups of all ambients that can be crossed by $m[P]$, and the set $\mathbf{H}$ includes the groups of all ambients that can be opened within $m[P]$.

A corollary of Theorem 3 is that these bounds on ambient behavior apply not just to ambients contained within Q, but to ambients contained in any process reachable by a series of reductions from Q.

6 Conclusions

Our contribution is a new type system for tracking the behavior of mobile computations. We introduced the idea of a *name group*. A name group represents a collection of ambient names; ambient names belong to name groups in the same sense that values belong to types. We studied the properties of a new process operator $(\nu G)P$ that lexically scopes groups. Using groups, our type system can impose behavioral constraints like "this ambient crosses only ambients in one set of groups, and only dissolves ambients in another set of groups". Our previous type system for mobility [CGG99] cannot express such constraints.

In the extended version of this paper [CGG00], we revisit an encoding of a distributed programming language that we first reported in the technical report version of our earlier work [CGG99]. In the encoding, ambients model both network nodes and the threads that may migrate between the nodes. The encoding can be typed in all three of the systems presented in this paper. The encoding illustrates how ambient groups can be used to partition the set of ambient names according to their intended usage, and how opening and crossing control allows the programmer to state some of those programming invariants which are the most interesting when programming mobile computation. For example, the typing allows threads to cross node boundaries, but not mistakenly the other way round, and guarantees that neither threads nor nodes may be opened. We use (νG) to make fresh groups for certain synchronization ambients in the encoding. The benefit of (νG) is that we can be statically assured that these synchronization ambients are known only to the processes we intend to synchronize, and propagate no further.

Our groups are similar to the *sorts* used as static classifications of names in the π-calculus [Mil99]. Our basic system of Section 2 is comparable to Milner's sort system for π, except that a *new sort* operator does not seem to have been considered in the π-calculus literature. Another difference is that sorts in the π-calculus are mutually recursive; we would have to add a recursion operator to achieve a similar effect. Our systems of Sections 3 and 4 depend on groups to constrain the opening and crossing behavior of processes. We are not aware of any uses of Milner's sorts to control process behavior beyond controlling the sorts of communicated names.

Apart from Milner's sorts, other static classifications of names occur in derivatives of the π-calculus. We mention two examples. In the type system of Abadi [Aba97] for the spi calculus, names are classified by three static *security levels—Public*, *Secret*, and *Any*—to prevent insecure information flows. In the flow analysis of Bodei, Degano, Nielson, and Nielson [BDNN98] for the π-calculus, names are classified by static *channels* and *binders*, again with the purpose of establishing security properties. (A similar flow analysis now exists for the ambient calculus [NNHJ99].) Although there is a similarity between these notions and groups, and indeed to sorts, nothing akin to our (νG) operator appears to have been studied.

There is a connection between name groups and the region variables in the work of Tofte and Talpin [TT97] on region-based implementation of the λ-calculus. The store is split into a set of stack-allocated regions, and the type of each stored value is labelled with the region in which the value is stored. The scoping construct *letregion* ρ *in* e allocates a fresh region, binds it to the region variable ρ, evaluates e, and on completion, deallocates the region bound to ρ. The constructs *letregion* ρ *in* e and $(\nu G)P$ are similar in that they confer static scopes on the region variable ρ and the group G, respectively. One difference is that in our operational semantics $(\nu G)P$ is simply a scoping construct; it allocates no storage. Another is that scope extrusion laws do not seem to have been explicitly investigated for *letregion*. Still, we can interpret *letregion* in terms of (νG), and intend to report this in a future paper.

Levi and Sangiorgi's type system for a generalization of the ambient calculus [LS00] can guarantee immobility and single-threadedness. It would be interesting to consider extensions of their type system with groups.

Acknowledgements Silvano Dal Zilio commented on a draft of this paper. Ghelli acknowledges the support of Microsoft Research during the writing of this paper. The same author has also been partially supported by grants from the E.U., workgroups PASTEL and APPSEM, and by "Ministero dell'Università e della Ricerca Scientifica e Tecnologica", project INTERDATA.

References

[Aba97] M. Abadi. Secrecy by typing in security protocols. In *Proceedings TACS'97*, *LNCS* 1281, pages 611–638. Springer, 1997.

[BDNN98] C. Bodei, P. Degano, F. Nielson, and H. Nielson. Control flow analysis for the π-calculus. In *Proceedings Concur'98*, *LNCS* 1466, pages 84–98. Springer, 1998.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings FoSSaCS'98*, *LNCS* 1378, pages 140–155. Springer, 1998. Accepted for publication in *Theoretical Computer Science*.

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings POPL'99*, pages 79–92. ACM, 1999.

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *Proceedings ICALP'99*, *LNCS* 1644, pages 230–239. Springer, 1999.

[CGG00] L. Cardelli, G. Ghelli, and A. D. Gordon. Types for the ambient calculus. Microsoft Research Technical Report, to appear.

[GC99] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *Proceedings FoSSaCS'99*, *LNCS* 1578, pages 212–226. Springer, 1999.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings POPL'00*, pages 352–364. ACM, 2000.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.

[NNHJ99] F. Nielson, H.R. Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in mobile ambients. In *Proceedings Concur'99*, *LNCS* 1664, pages 463–477. Springer, 1999.

[TT97] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997. Preliminary version in *Proceedings POPL'94*.

An Asynchronous, Distributed Implementation of Mobile Ambients

Cédric Fournet[1], Jean-Jacques Lévy[2], and Alan Schmitt[2]

[1] Microsoft Research
[2] INRIA Rocquencourt

Abstract We present a first distributed implementation of the Cardelli-Gordon's ambient calculus. We use Jocaml as an implementation language and we present a formal translation of Ambients into the distributed join calculus, the process calculus associated with Jocaml. We prove the correctness of the translation.

1 Introduction

We present a highly concurrent distributed implementation of the Cardelli-Gordons's calculus of Mobile Ambients [4] in Jocaml [13,6]. The ambient calculus is a simple and very esthetic model for distributed mobile computing. However, until now, it did not have a distributed implementation. Such an implementation may seem easy to build, especially with a language with distribution and strong migration (Jocaml), but we encountered several difficulties and design choices.

Ambients are nested. Their dynamics is defined by three atomic steps: an ambient may move into a sibling ambient (In), it may move out of its parent ambient (Out), or it may open one of its child ambients (Open). Each atomic migration step may involve several ambients, possibly on different sites. For instance, the source and destination ambients participate to an In-step; similarly the source and parent ambients take part to an Out-step; the target ambient participates to an Open-step. Each atomic step of the ambients calculus can be decomposed in two parts: checking whether the local structure of the ambient tree enables the step, and actually performing the migration.

The first part imposes some distributed synchronization. One may use a global synchronous primitive existing at the operating system or networking level, but such a solution is unrealistic in large-scale networks. A first range of solutions can be designed by considering locks and critical sections in order to serialize the implementation of atomic steps. For instance, the two ambients participating to a reduction step are temporary locked, and the locks are released at the end of the step. However this solution cannot be symmetric, in the same way as there is no symmetric distributed solution to the Dining Philosophers problem. Some ambients have to be distinguished, for instance, one ambient could be the synchronizer of all ambients. Naturally, the nested structure of ambients can be used, for instance each ambient controls the synchronization of its direct subambients. In both cases, one has to be careful to avoid deadlocks or

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 348–364, 2000.
© Springer-Verlag Berlin Heidelberg 2000

too much serialization. This solution would be similar to Cardelli's centralized implementation of an earlier variant of the ambient calculus in Java [1,2]. One advantage of a serialized solution is the ease of the correctness proof of the implementation. On the negative side, each attempt to perform a step takes several locks higher up in the ambient hierarchy; these locks may be located at remote sites, leading to long delays before these locks are released for other local steps. Moreover, due to the mobility discipline of the ambient calculus, an ambient that migrates from one point to another in the ambient hierarchy has to travel through an ambient enclosing both the origin and the destination, thus inducing global bottlenecks.

A different set of solutions is fully asynchronous. Atomic steps of ambients are decomposed into several elementary steps, each involving only local synchronization. In this approach, each ambient step is implemented as a run of a protocol involving several messages. Concurrency is higher, as only the moving ambient might not be available for other reduction steps. For instance, our solution never blocks steps involving parents of a moving ambient. The implementation of migration towards a mobile target may be problematic, but can be handled independently of the implementation of ambient synchronization, e.g., using a forwarding mechanism. In our case, we simply rely on the strong migration primitive of Jocaml. On the negative side, the correctness proof is more involved.

In this paper, we present an asynchronous distributed algorithm for implementing ambients, we make it precise as a translation into the join calculus—the process calculus that is a model of Jocaml [9], and we refine this translation into a distributed implementation of ambients written in Jocaml. The algorithm provides an insight into the implementability of ambients. The Jocaml prototype is a first, lightweight, distributed implementation of ambients. The translation is proved correct in two stages: first we use barbed coupled simulations for the correctness of the algorithm, then we use an hybrid barbed bisimulation for the actual translation into the join calculus. Technically, the first stage is a first application of coupled-simulations [17] in a reduction-based, asynchronous setting; it relies on the introduction of an auxiliary ambient calculus extended with transient states; it does not depend of the target language. The second stage is a challenging application of the decreasing diagram technique [16]. In combination, these results imply that the translation preserves and reflects a variety of global observation predicates.

The paper is organized as follows. In section 2, we present the asynchronous algorithm and we show a formal translation from ambient processes to join calculus processes. In section 3, we discuss the correctness of the translation in terms of observations. In section 4, we focus on the operational correspondence between a process and its translation; to this end, we refine the ambient calculus to express additional transient states induced by the translation. In section 5, we state our main technical results and give an idea of their proofs. In section 6, we describe more practical aspects of the Jocaml implementation. We conclude in section 7. In an appendix, we recall the operational semantics of the distributed

join calculus and of the calculus of mobile ambients, and we give an overview of both calculi. Additional discussions, technical details, and all the proofs appear in the full version of this paper [10].

2 From Ambients to the Join Calculus

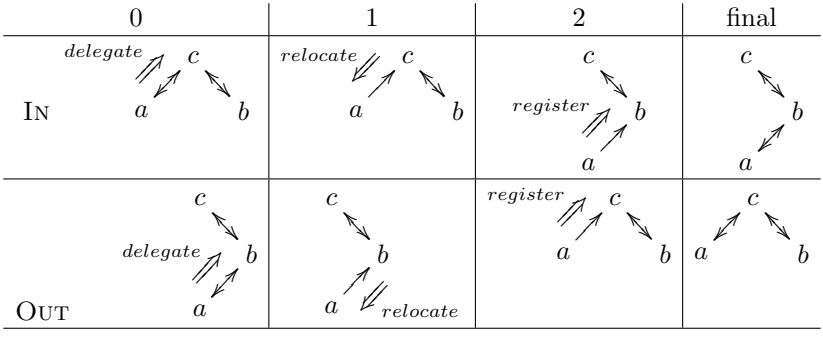
We describe the asynchronous algorithm, then we specify it as a translation from ambients to the join calculus. We begin with a fragment of the ambient calculus given by the grammar $P ::= a[\,P\,] \mid \texttt{in}\ a.P \mid \texttt{out}\ a.P \mid P \mid P' \mid \mathbf{0}$. In a second stage, we incorporate Open steps and other ambient constructs.

2.1 An Asynchronous Algorithm

The dynamic tree structure of ambients is represented by a doubly linked tree. Each node in the tree implements an ambient: each node contains non-ambient processes such as $\texttt{in}\ b.P$ or $\texttt{out}\ a.c[Q]$ running in parallel; each node also hosts an *ambient manager* that controls the steps performed in this ambient and in its direct subambients. Different nodes may be running at different physical sites, so their ambient managers should use only asynchronous messages to communicate with one another. Since several ambients may have the same name, each node is also associated with a unique identifier. (Informally, we still refer to ambients by name, rather than unique identifier.)

Each ambient points to its subambients and to its parent ambient. The down links are used for controlling subambients, the up link is used for proposing new actions. The parent of the moving ambient for an In-step knows the destination ambient; the parent also knows the destination ambient—its own parent—for an Out-step; it controls the opened ambient for an Open-step. Hence, the decision to perform a step will always be taken by the parent of the affected ambient.

Moves of ambient a in and out of ambient b correspond to three successive steps, depicted below. Single arrows represent current links; double arrows represent messages in transit.



We detail the dynamics of an In-step, e.g., $c[\,a[\,\texttt{in}\ b.Q\,]\ \mid\ b[\,\mathbf{0}\,]\,] \to c[\,b[\,a[\,Q\,]\,]\,]$.

0**-step:** initially, a delegates the migration request In b to its current parent (here c); to this end, it uses its current up link to send a message to c saying that a is willing to move into an ambient named b.

1**-step:** the enclosing ambient c matches a's request with a's and b's down links. Atomically, a's request and the down link to a are erased, and a relocation message is sent to a; this message contains the address of b, so that a will be able to relocate to b, and also a descriptor of a's successful action, so that a can complete this step by triggering its guarded process Q.

2**-step:** the moving ambient a receives c's relocation message, relocates to b's site, and updates its up link to point to b. It also sends a message to b that eventually registers a as a subambient of b, establishing the new downlink.

The 1-step may preempt other actions delegated by a to its former parent c. Such actions should now be delegated to its new parent b. For that purpose, a's ambient manager keeps a log of the pending actions delegated in 0-steps, and, as it completes one of these actions in a 2-step, it re-delegates all other actions towards its new parent. The log cannot be maintained by the parent, because delegation messages may arrive long after a's departure. Moreover, in the case an ambient moves back into a former parent, former delegation messages may still arrive, and should not be confused with fresh ones. Such stale messages must be deleted. This is not directly possible in an asynchronous world, but equivalently each migration results in a modification of the unique identifier of the moving ambient, each delegation message is tagged with the current identifier, and the parent discards every message with an old identifier.

An Out-step of a out of b corresponds to the same series of three steps. The main different is in step 1, as the enclosing ambient b matches a's request with a's down link and its own name b, and passes its own up link in the relocation message sent back to a.

2.2 A Simple Translation

The compositional translation $[\![\cdot]\!]_e$ appears in Figure 1. Overall, the tree of nested ambients is mapped to an isomorphic tree of nested locations. Each ambient is mapped to a join calculus location containing the definition D of the channel names that form the ambient interface, and containing processes that represent the ambient state. The definition D is composed of three groups of rules D_0, D_1, and D_2 that respectively implement 0, 1, and 2-steps of the algorithm.

To represent the distributed data structure used in the algorithm of section 2.1, an ambient is represented by an interface e, which is a record that contains fields *here*, *amb*, sub_{in}, sub_{out}, *reloc*, *in*, and *out*. The *here*-field is the name of the location hosting the ambient, whereas the other fields are channel names used to interact with this ambient. The translation is parameterized by the interface e of the current enclosing ambient. A down link to a subambient named b with interface e_b and unique identifier (uid) j is represented as a message $amb(j, b, e_b)$. For every ambient, the up link to its parent ambient is represented by the parent interface e, which is stored in the state message $s(a, i, e, l)$. In

$$
\begin{aligned}
[\![a[\,P\,]]\!]_e &= \texttt{def}\ AM_{a,e}(P)\ \texttt{in}\ \mathbf{0} \\
[\![\texttt{in}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.in(a,\kappa) \\
[\![\texttt{out}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.out(a,\kappa) \\
[\![P \mid Q]\!]_e &= [\![P]\!]_e \mid [\![Q]\!]_e \\
[\![\mathbf{0}]\!]_e &= \mathbf{0}
\end{aligned}
$$

where the ambient manager $AM_{a,e}(P)$ is defined as:

$$
\begin{aligned}
D_0 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid in(b,\kappa) \ \triangleright s(a,i,e,l \cup \{\text{In } b\ \kappa\}) \mid e.sub_{in}(i,b,\kappa) \\
&\quad \wedge\ s(a,i,e,l) \mid out(b,\kappa) \triangleright s(a,i,e,l \cup \{\text{Out } b\ \kappa\}) \mid e.sub_{out}(i,b,\kappa) \\
D_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid amb(k,c,e_c) \mid sub_{in}(k,b,\kappa) \triangleright \\
&\qquad s(a,i,e,l) \mid amb(j,b,e_b) \mid e_c.reloc(e_b,\kappa) \\
&\quad \wedge\ s(a,i,e,l) \mid amb(j,b,e_b) \mid sub_{out}(j,a,\kappa) \triangleright s(a,i,e,l) \mid e_b.reloc(e,\kappa) \\
D_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid reloc(e',\kappa) \triangleright \texttt{go}(e'.here); (I_{a,e_h,e'} \mid \kappa() \mid Flush(l,in,out,\kappa)) \\
D &\stackrel{\text{def}}{=} D_0 \wedge D_1 \wedge D_2
\end{aligned}
$$

$$
I_{a,e_h,e} \stackrel{\text{def}}{=} \texttt{def uid}\ i\ \texttt{in}\ s(a,i,e,\emptyset) \mid e.amb(i,a,e_h)
$$

$$
AM_{a,e}(P) \stackrel{\text{def}}{=} here\left[D\ :\ I_{a,e_h,e} \mid [\![P]\!]_{e_h}\right]
$$

with the record notation $e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, \\ sub_{out} = sub_{out}, reloc = reloc, in = in, out = out \end{array} \right\}$.

Figure 1. Translation from In/Out ambient processes to the join calculus

addition, the state message contains the name a and the current uid i of the ambient, and the log l of In and Out actions that have been delegated to the parent ambient using e.

We resume our study of an In-action, considering the role of each message in the translation of $c[\,a[\,\texttt{in}\ b.Q\,]\ \mid\ b[\,\mathbf{0}\,]\,]$. Initially, the translation defines a continuation κ for Q and issues a message $in(b,\kappa)$ in a, which is a subjective migration request into an ambient named b.

The 0-step consists of delegating the request to the parent ambient. Using the first rule of a's D_0, the messages $s(a,i,e,l)$ and $in(b,\kappa)$ are consumed, the request is recorded in a's log as an entry In $b\ \kappa$, and the request is forwarded to the enclosing ambient c described by the interface e. The ambient a remains active, with new state $s(a,i,e,l \cup \{\text{In } b\ \kappa\})$. In parallel, the message $e.sub_{in}(i,b,\kappa)$ is a subambient move request sent to c, with the explicit identifier i of the requester a.

The 1-step is performed by c's ambient manager. The message $sub_{in}(i,b,\kappa)$ may be consumed using the first rule of D_1. The rule also requires that both the ambient that issued the request and another destination ambient with name b be actually present. This step removes the down link for the moving ambient—the message $amb(i,a,e_a)$—, thus blocking other actions competing for the same message, whereas the destination ambient remains available for concurrent steps. A

$$
\begin{aligned}
[\![\texttt{open}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.open(a,\kappa)\\
[\![\langle n\rangle]\!]_e &= e.send(n)\\
[\![(n).P]\!]_e &= \texttt{def}\ \kappa(n) \triangleright [\![P]\!]_e\ \texttt{in}\ e.recv(\kappa)\\
[\![!P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e \mid \kappa()\ \texttt{in}\ \kappa()\\
[\![\nu a.P]\!]_e &= \texttt{def}\ \texttt{fresh}\ a\ \texttt{in}\ [\![P]\!]_e
\end{aligned}
$$

with additional rules in the definition of $AM_{a,e}(P)$:

$$
\begin{aligned}
D_1' &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid open(b,\kappa) \triangleright s(a,i,e,l) \mid e_b.opening(\kappa)\\
D_2' &\stackrel{\text{def}}{=} s(a,i,e,l) \mid opening(\kappa) \triangleright f(e) \mid \kappa() \mid Flush(l, e.in, e.out, \kappa)\\
D_C &\stackrel{\text{def}}{=} s(a,i,e,l) \mid recv(\kappa) \mid send(n) \triangleright s(a,i,e,l) \mid \kappa(n)\\
D_F &\stackrel{\text{def}}{=} f(e) \mid in(b,\kappa) \triangleright f(e) \mid e.in(b,\kappa)\\
&\quad \wedge\ f(e) \mid out(b,\kappa) \triangleright f(e) \mid e.out(b,\kappa)\\
&\quad \wedge\ f(e) \mid open(b,\kappa) \triangleright f(e) \mid e.open(b,\kappa)\\
&\quad \wedge\ f(e) \mid amb(j,b,e_b) \triangleright f(e) \mid e.amb(j,b,e_b)\\
&\quad \wedge\ f(e) \mid sub_{in}(k,b,\kappa) \triangleright f(e) \mid e.sub_{in}(k,b,\kappa)\\
&\quad \wedge\ f(e) \mid sub_{out}(k,b,\kappa) \triangleright f(e) \mid e.sub_{out}(k,b,\kappa)\\
&\quad \wedge\ f(e) \mid recv(\kappa) \triangleright f(e) \mid e.recv(\kappa)\\
&\quad \wedge\ f(e) \mid send(b) \triangleright f(e) \mid e.send(b)
\end{aligned}
$$

$$
D \stackrel{\text{def}}{=} D_0 \wedge D_1 \wedge D_1' \wedge D_2 \wedge D_2' \wedge D_C \wedge D_F
$$

with the extended record notation

$$
e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, sub_{out} = sub_{out}, open = open,\\ reloc = reloc, in = in, out = out, opening = opening, recv = recv, send = send \end{array} \right\}
$$

Figure 2. Additional clauses for the full translation

relocation message $e_a.reloc(e_b, \kappa)$ is emitted, signalling to the requesting ambient a that it must migrate to the ambient with interface e_b, with continuation κ.

The 2-step, using a's rule D_2, consumes the message on $reloc$ and the current state message, performs a join calculus migration to the location of the destination ambient, then resumes the activities of a with parent interface e_b. To this end, the process I_{a,e_a,e_b} restores an active state: it generates a fresh uid i', issues a local state message $s(a, i', e_b, \emptyset)$ representing the up link, and sends to the new parent a message $e_b.amb(i', a, e_a)$ representing the down link. (Since no down link will ever mention the previous uid i, previous delegation messages tagged with i will never match a rule of D_1. In our implementation, these stale messages are actually discarded.) In addition, the message $\kappa()$ triggers the continuation. Finally, the process $Flush(l \cup \{\text{In}\ b\ \kappa\}), in, out, \kappa)$ restarts any preempted actions appearing in the log. As defined in the appendix, this process emits a message $in(d, \kappa')$ or $out(d, \kappa')$ for every entry In d κ' or Out d κ' appearing in the log l and such that $\kappa' \neq \kappa$. These entries correspond to actions preempted by the

migration; they will be delegated to a new parent through other iterations of 0-steps.

Similarly, an OUT-step is performed according to the algorithm by using the second rule of D_0 of the moving ambient, the second rule of D_1 of the enclosing ambient, and finally rule D_2 of the moving ambient.

2.3 Dealing with Other Ambient Constructs

The translation of Figure 2 generalizes the translation above to the full ambient calculus. For each additional construct, we add a clause to the compositional translation $[\![\cdot]\!]_e$. We also upgrade $AM_{a,e}(\cdot)$, and use a larger environment e with extra fields for *open*, *opening*, *recv*, and *send*.

Values and Scopes. Ambient names are mapped to identical names in the join calculus. The two calculi rely on similar lexical scope disciplines, with scope extrusion performed by structural equivalence (rule SCOPE in join, rules R1 and R2 in ambients). Thus, it suffices to translate the creation of local ambient names $\nu a.P$ into binders `fresh` a with the same scope in the join calculus.

Communication. Ambient communication is implemented by supplementing every ambient manager with a rule D_C that binds two channels *send* and *recv* and synchronizes message outputs and message requests. This encoding is much like the encoding of pi-calculus channels into the join calculus (see [8]).

Replication. Each replicated process $!P$ is coded using a standard recursive encoding of infinite loops in the join calculus.

Open. Ambient processes may dissolve ambient boundaries using the `open` capabilities. In contrast, join calculus names are statically attached to their defining location, and location boundaries never disappear. We thus lose the one-to-one mapping from ambients to locations, and distinguish two states for each location: either the ambient is still running and the message s is present, or it has been opened and henceforth messages sent to its interface are forwarded to the enclosing ambient. The indirection is achieved by using a persistent message sent on f defined in D_F. This leads to complications in the proofs, as one must prove that these opened locations do not interfere with the rest of the translation.

3 Correctness of the Translation

The distributed synchronization algorithm seems to depart from the operational semantics of ambients, and the translation of nested ambients yields arbitrarily large terms with numerous instances of the algorithm running in parallel. This makes the correctness of the distributed implementation problematic. Technically, both calculi have a reduction-based semantics, which can be equipped with standard notions of observation. This provides a precise setting for establishing correctness on the translation, rather than on an abstraction of the algorithm.

(Of course, there are still minor discrepancies between the translation and the actual code in Jocaml; see section 6.)

We first define a syntactic notion of observation. For each calculus, we use a family of predicates on processes P indexed by names b, written $P \downarrow_b$.

- In the ambient calculus, $P \downarrow_b$ when b is free in P and $P \equiv \nu\widetilde{v}.(\, b[\, Q\,] \mid R)$.
- In the join calculus, $P \downarrow_b$ when b is free in P and $P \equiv \alpha[D' : b(\widetilde{v}) \mid P'] \wedge D$.

Next, we express the correctness of the translation in terms of the following predicates, for both ambient and join processes:

- A process P has a *weak barb* on b (written $P \Downarrow_b$) when $P \rightarrow^* P' \downarrow_b$.
- A process P *diverges* (written $P \Uparrow$) when P has an infinite series of steps.
- A process P has a *fair-must barb* on b (written $\Box P \Downarrow_b$) when for all P' such that $P \rightarrow^* P'$, we have $P' \Downarrow_b$.

In combination, these predicates give a precise content to the informal notion of correctness: "the translation should neither suppress existing behaviors, nor introduce additional behaviors." The minimal notion of correctness for an implementation is the reflection of weak barbs, which rules out spurious behaviors; the converse direction states that the implementation does not discard potential behaviors; for instance, it rules out new deadlocks, or even an empty translation. The preservation of convergence is of pragmatic importance. In addition, correctness for fair-must tests relates infinite computations [7], and rules out implementations with restrictive scheduling policies.

Since top-level ambients are not translated into messages on free names, the observation of translated ambients requires some special care. To this end, we supplement the translation at top-level with a definition D_t that reveals the presence of a particular barb $\downarrow_b$ in the source process. Using D and e_h as defined in figure 2, and for a given interface e, we define the top-level translation

$$[\![P]\!]^b \stackrel{\text{def}}{=} here[D \wedge D_t \wedge \texttt{uid}\ i : s(a, i, e, \emptyset) \mid t(b) \mid [\![P]\!]_{e_h}]$$
$$D_t \stackrel{\text{def}}{=} s(a, i, e, l) \mid amb(j, b, e_b) \mid t(b) \triangleright s(a, i, e, l) \mid amb(j, b, e_b) \mid yes()$$

Without loss of generality, we always assume that the names in a, i, t, e, yes do not clash with any name free in P, and that the location and channel names introduced by the translation do not clash with any name of P.

We are now ready to state that all the derived observations discussed above are preserved and reflected by the translation:

Theorem 1. *For every ambient process P and name b, we have $P \Downarrow_b$ if and only if $[\![P]\!]^b \Downarrow_{yes}$; $P \Uparrow$ if and only if $[\![P]\!]^b \Uparrow$; and $\Box P \Downarrow_b$ if and only if $\Box[\![P]\!]^b \Downarrow_{yes}$.*

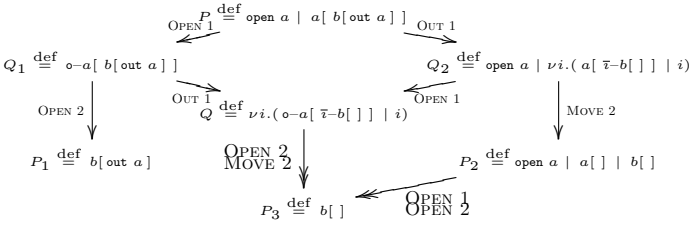
While correctness is naturally expressed in terms of observations along the reduction traces of processes, its proof is challenging. In particular, a direct approach clearly leads to intractable inductions on both the syntax of the source process and on the series of reductions.

4 A Calculus of Ambients Extended with Transient States

In order to prove theorem 1, we introduce an intermediate calculus of ambients with constructs that materialize the key transient states of the algorithm of section 2.1, and we equip this calculus with a reduction semantics in direct correspondence with the algorithm. For instance, atomic In steps are decomposed into series of 1- and 2-steps. (However, 0-steps are not represented, inasmuch as requests are always eventually delegated to the current parent.) In the next section, we rely on the extended calculus to establish correctness as the composition of two equivalences. First, we use *coupled simulations* [17] to relate the two semantics for ambients; then, we use bisimulations to relate ambients equipped with the extended semantics to their join calculus translations.

The grammar for the extended calculus appears in Figure 3. It has new processes representing ambients that are committed to move or to be opened, as the result of their father's 1-step—we call such transient ambients *stubs*—and also new processes marking the future position of migrating ambients—we call such precursors *scions*. Pairs of stubs and scions are syntactically connected by a marker i. The extended operational semantics appears in Figure 4. It is a reduction-based semantics with auxiliary labels for stubs and scions. Each of the reduction steps In, Out, and Open is decomposed in two steps. Initial steps $\rightarrow_1$ introduce stubs and scions; completion steps $\rightarrow_2$ consume them. The reduction rules for Recv and Repl are those of the original semantics, except that we write $\rightarrow_C$ instead of $\rightarrow$. Overall, we obtain a reduction system for extended ambient processes with steps $\rightarrow_{12C} \stackrel{\text{def}}{=} \rightarrow_1 \cup \rightarrow_2 \cup \rightarrow_C$. For the sake of comparison, we also extend the original reduction semantics and the observation predicates from ambients to extended ambients.

Initially, stubs and scions are neighbors, but they may drift apart as the result of other steps before performing the matching completion step, so an auxiliary labeled transition system is used to match stubs and scions. The completion of a deferred migration is thus rendered as a global communication step between processes residing in two different ambient contexts, and syntactically linked by the name i. This may seem difficult to implement, but actually scions have no operational contents; they represent the passive target for a strong migration. To illustrate the extended calculus, we give below the reductions for a process with a critical pair. Processes P_i are regular ambient processes; processes Q_i are transient processes reflecting intermediate states of our algorithm.



$P ::=$		extended ambient process
	$\ldots$	*all the constructors of Figure 5*
	$\mid\ Xn[\,P\,]$	stub
	$\mid\ i$	scion
	$\mid\ \nu i.P$	marker restriction
$X ::=$		state extension
	$\bar{\imath}\{P\}$–	the stub is committed to move to i
	$\mid\ \mathtt{o}\{P\}$–	the stub is being opened

Well-formed conditions: stubs and scions may occur only in extended evaluation contexts; restricted markers i must be used linearly (exactly one stub and one scion). In the following, we write $X^{=}n[P]$ for either $Xn[P]$ or $n[P]$.

Figure 3. Syntax for an ambient calculus extended with transient states

Extended evaluation contexts $E(\cdot)$ are defined by the grammar

$$E(\cdot) ::= \ \cdot \ \mid\ P \mid E(\cdot) \ \mid\ E(\cdot) \mid P \ \mid\ X^{=}n[\,E(\cdot)\,] \ \mid\ \nu n.E(\cdot) \ \mid\ \nu i.E(\cdot)$$

Structural equivalence $\equiv$ is the smallest equivalence relation on processes that is closed by application of extended evaluation contexts and by α-conversion, and that satisfies the axioms P0, P1, P2, C0, C1, R1, R2 of figure 6 and:

$$\text{R2X}\ \ \frac{m \neq n \quad m \text{ is not free in } X}{Xn[\nu m.P] \equiv \nu m.Xn[P]}$$

Labeled transitions $\xrightarrow{\alpha}$ are the smallest families of relations closed by application of restriction-free extended evaluation contexts and such that

$$\textsc{Stub}\ \ \bar{\imath}\{Q\}\text{–}n[R] \xrightarrow{\bar{\imath}.n[Q|R]} \mathbf{0} \qquad \textsc{Scion}\ \ i \xrightarrow{i.P} P$$

Original reduction steps $\rightarrow$ are defined as in Figure 6 with extended evaluation contexts. *Initial steps* $\rightarrow_1$*, Completion steps* $\rightarrow_2$*, and Other steps* $\rightarrow_C$ are the smallest relations closed by structural equivalence, by application of extended evaluation contexts, and such that:

$$\textsc{In 1}\ \ \begin{array}{c} m[P] \mid n[\mathtt{in}\ m.Q \mid R] \\ \rightarrow_1 \nu i.\ m[i \mid P] \mid \bar{\imath}\{Q\}\text{–}n[R] \end{array} \qquad \textsc{Out 1}\ \ \begin{array}{c} X^{=}m\big[P \mid n[\mathtt{out}\ m.Q \mid R]\big] \\ \rightarrow_1 \nu i.\ i \mid X^{=}m\big[P \mid \bar{\imath}\{Q\}\text{–}n[R]\big] \end{array}$$

$$\textsc{Move 2}\ \ \frac{P \xrightarrow{\bar{\imath}.S} P' \quad Q \xrightarrow{i.S} Q'}{\nu i.(P \mid Q) \rightarrow_2 P' \mid Q'}$$

$$\textsc{Open 1}\ \ \mathtt{open}\ n.Q \mid n[R] \rightarrow_1 \mathtt{o}\{Q\}\text{–}n[R] \qquad \textsc{Open 2}\ \ \mathtt{o}\{Q\}\text{–}n[R] \rightarrow_2 Q \mid R$$

$$\textsc{Recv}\ \ \langle n\rangle \mid (x).P \rightarrow_C P\{{}^{n}/_{x}\} \qquad \textsc{Repl}\ \ !P \rightarrow_C P \mid !P$$

Figure 4. Semantics for an ambient calculus extended with transient states

5 Coupled Simulations and Operational Correspondence

We continue our study of correctness in terms of equivalences based upon weak barbs. These equivalences are essential to obtain a modular proof. As a side benefit, they also provide a finer account of correctness. (See [7] for a discussion of equivalences and encodings in process calculi.) Instead of equivalences, we actually often use relations ranging over different domains, equipped with different notions of reduction steps $\rightarrow_a$, $\rightarrow_b$ and families of observations $\Downarrow_{a,x}$ and $\Downarrow_{b,x}$.

Definition 1 (Barbed bisimulations). *A relation $\mathcal{R} \in \mathcal{P}_a \times \mathcal{P}_b$ is a weak barbed simulation when, for all $P\ \mathcal{R}\ Q$, we have (1) if $P \rightarrow_a^* P'$, then there exists Q' such that $Q \rightarrow_b^* Q'$ and $P'\ \mathcal{R}\ Q'$; (2) for all x, if $P \Downarrow_{a,x}$, then $Q \Downarrow_{b,x}$. $\mathcal{R}$ is a barbed bisimulation when $\mathcal{R}$ and its converse $\mathcal{R}^{-1}$ are barbed simulations.*

Bisimulations come with effective proof techniques that consider only a few steps at a time, rather than whole execution traces. Unfortunately, barbed bisimilarity $\approx$—the largest barbed bisimulation closed by application of evaluation contexts—is too discriminating for our protocol. Transient processes such as Q_1 in the example above account for a partially-committed internal choice: Q_1 may reduce to P_1 and P_3, but not to P_2. In some contexts, they are not bisimilar to any derivative of P in the original ambient semantics. To address this issue of gradual commitment, Parrow and Sjödin proposed coarser relations called coupled simulations [17,15]. We liberally adapt their definition to ambients:

Definition 2 (Coupled simulations). *The relations $\leqslant \in \mathcal{P}_a \times \mathcal{P}_b$ and $\leqslant \in \mathcal{P}_b \times \mathcal{P}_a$ form a pair of* barbed coupled simulations *when $\leqslant$ and $\leqslant$ are barbed simulations that meet the coupling conditions: (1) if $P \leqslant Q$, then $Q \rightarrow_b^* \leqslant P$; (2) if $Q \leqslant P$, then $P \rightarrow_a^* \leqslant Q$.*

The discrepancy between $\leqslant$ and $\geqslant$ is most useful for handling transient states such as Q_1. The coupling conditions guarantee that every transient state can be mapped both to a less advanced state and to a more advanced one. In our case, we would have $Q_1 \leqslant P$ and $P_i \leqslant Q_1$ for $i = 1, 3$.

Correctness of the Asynchronous Algorithm The first stage of our correctness argument is expressed as coupled simulations between ambient processes equipped with the original and the extended semantics. In the statement below, related processes have the same syntax, but live in different calculi, equipped with different reduction semantics.

Theorem 2. *Let $\lesseqgtr$ be the union of $\leqslant \cap \geqslant$ for all barbed coupled simulations between ambient and extended ambient processes that are closed by application of evaluation contexts. For all ambient processes P, we have $P \lesseqgtr P$.*

The proof appears in [10]; it makes apparent some subtleties of our algorithm due to additional concurrency. After a first series of results on partial commutation properties for extended steps, the key lemmas establish that, for any ambient process P and extended ambient process Q, if $P \rightarrow_{12C}^* Q$, then (1) for some ambient process P' we have $Q \rightarrow_{12C}^* P'$ and (2) for any such process P', we also have $P \rightarrow^* P'$ in the original semantics.

Operational Correspondence The second stage of the proof relates ambients equipped with the extended semantics to their join calculus translations. It is simpler than the first one, in principle, but its proof is complicated because the translation makes explicit many details of the implementation that are inessential to the algorithm.

In order to express the correspondence of observations across the translation, we supplement the top-level translation $[\![\cdot]\!]^b$ of theorem 1 with an external choice of the ambient barb to be tested. With the same notations, we write $[\![\cdot]\!]^t$ for the translation that maps every process P to the process $[D \wedge D_t \wedge \texttt{uid}\ i : s(a, i, e, \emptyset) \mid p(t) \mid [\![P]\!]_{e_0}]$. As before, we assume that names in a, i, e, p, t, and *yes* do not clash with names free in P. At any point, we can use the evaluation context $T_b(\cdot) \stackrel{\text{def}}{=} h_T[p(t) \triangleright t(b) : \mathbf{0}] \wedge (\cdot)$ to test a translated ambient barb on b by testing the plain join calculus barb $T_b(\cdot) \Downarrow_{yes}$. We have $[\![P]\!]^b \approx T_b([\![P]\!]^t)$ in the join calculus.

Theorem 3 (Correctness of the translation). *Let $\approx^{aj}$ be the largest bisimulation between extended ambient processes with reductions $\rightarrow_{12C}$ and join processes such that $Q \approx^{aj} R$ implies $Q \Downarrow_b$ iff $T_b(R) \Downarrow_{yes}$. For all ambient processes P, we have $P \approx^{aj} [\![P]\!]^t$.*

In the long version of the paper [10], a more precise *strong bisimulation up to bookkeeping* result is proved for the translations of all reachable extended ambient processes. Since every significant transient state induced by the translation has been lifted to the extended ambient calculus, its proof essentially amounts to an operational correspondence between the two calculi. We partition reductions in the join calculus according to the static rule of the translation being used. For instance, we let $\rightarrow_1$ steps in the join calculus be the steps using a rule of a definition $D_1 \wedge D_1'$ of figure 2; these steps create a *reloc* or an *opening* message, and are in operational correspondence with source $\rightarrow_1$ steps. We obtain two main classes of join calculus steps: steps $\rightarrow_{12C}$ that can be traced back to extended ambient steps, and "bookkeeping" steps $\rightarrow_B$, which are auxiliary steps used to trigger continuations, migrate, manage the logs, or unfold new ambient managers.

The main lemmas describe dynamic simplifications of derivatives of the translation, which are required to obtain translations of derivatives in the extended source calculus. These lemmas are expressed as elementary commutation diagrams between simplification relations and families of reduction steps. For instance, one lemma states that "stale messages" can be discarded; another, more complex lemma states that locations and ambient managers representing opened ambients can be eliminated, effectively merging the contents of opened ambients with the contents of their previously-enclosing ambient. To conclude, we exhibit a bisimulation relation between extended ambients equipped with steps $\rightarrow_{12C}$ and global translations of these extended ambients equipped with steps $\rightarrow_B^* \rightarrow_{12C} \rightarrow_B^*$. The proof is structured using the decreasing diagram technique of [16], whose conditions guarantee that every weak simulation diagram in the final proof can be obtained by gluing previously-established diagrams.

6 Distributed Implementation

In this section, we briefly describe the actual implementation in Jocaml. The initialization and the dynamics of distribution for ambient processes among several Jocaml runtimes lead to some design choice, discussed in the long version of the paper [10]. We also refer to [11] for the source code, setup instructions, and programming examples.

Our implementation closely follows the translation given in figures 1 and 2. Since Jocaml already provides support for mobility, local synchronization, and run-time distribution, our code is very compact—less than 400 lines for the interpreter, less than 40k in bytecode for the object files. The main differences between the formal translation and the code are given below:

- Messages in the implementation may pass names, but also arbitrary chains of capabilities [4], as for instance in the ambient process $\langle \texttt{in}\ a.\texttt{out}\ b \rangle\ |!(x).x.\langle x \rangle$.
- The implementation is an interpreter that delays the translation of guarded processes. The implementation maintains an environment for local variables, and performs dynamic type checking whenever a value is used either as a name or as a capability.
- The translation relies on non-linear join-patterns, which are not available in Jocaml. More explicitly, the implementation relies on hash tables to cache some messages: when a message arrives on sub_{in}, sub_{out}, *amb*, or *open*, either it is immediately used in combination with the message cache, or it is added to the cache.
- The formal translation of replication always yields a diverging computation (as in the source ambient process). More reasonably, the interpreter unfolds replication on demand: since every ambient reduction involves at most two copies of a replicated process, it suffices to initially unfold two copies, then to unfold an additional copy whenever a fresh copy is used or modified. Hence, the process $!\,a[\ ]$ does not diverge, while $!\,a[\,\texttt{in}\ a\,]$ still does.

7 Conclusions

We translated Mobile Ambients into the join calculus, and gave a first, asynchronous, distributed implementation of the ambient calculus in Jocaml, with a high level of concurrency. The synchronization mechanisms of Ambients turned out to be challenging first to implement, then to prove correct. This provides an insight into the ambient calculus as a model of concurrency. At the same time, this shows how Jocaml and its formal model can be used to tackle distributed and mobile implementations. For instance, the translation takes full advantage of join patterns to describe complex local synchronization steps, while a more traditional language would decompose these steps into explicit series of reads and updates protected by locks.

In order to get a safer and more efficient implementation, one should care about typing information for passed values and mobility capabilities [5,3]. Our implementation insures dynamic type-checking on values, whereas it would be

preferable to use the static type-checking discipline of Jocaml. Similarly, static knowledge of the actions that never appear in a given ambient can lead to more efficient, specialized ambient managers.

Finally, little is known about actual programming with Ambients, or the relevant abstractions to build a high-level language on top of the ambient calculus. While we did not consider changing the source language, we believe that our implementation provides an adequate platform for experimenting with ambient-based language design. For instance, our translation would easily accommodate the co-capabilities proposed in [14].

Acknowledgments. This work benefited from discussions with Luca Cardelli, Fabrice Le Fessant, and Luc Maranget.

References

1. L. Cardelli. Ambit, 1997. Available from http://www.luca.demon.co.uk/Ambit/Ambit.html.
2. L. Cardelli. Mobile ambient synchronization. Technical note 1997-013, Digital Systems Research Center, July 1997.
3. L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *ICALP'99*, volume 1644 of *LNCS*, pages 230–239, 1999.
4. L. Cardelli and A. Gordon. Mobile ambients. In *FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155, 1998.
5. L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99*, pages 79–92. ACM, Jan. 1999.
6. S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective-caml. In *ASA/MA'99*, pages 22–29. IEEE Computer Society, Oct. 1999.
7. C. Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, Ecole Polytechnique, Palaiseau, Nov. 1998. INRIA, TU-0556.
8. C. Fournet and G. Gonthier. The re exive chemical abstract machine and the join-calculus. In *POPL'96*, pages 372–385. ACM, Jan. 1996.
9. C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *CONCUR'96*, volume 1119 of *LNCS*, pages 406–421, Aug. 1996.
10. C. Fournet, J.-J. Lévy, and A. Schmitt. A distributed implementation of Ambients. Long version of this paper, available from http://join.inria.fr/ambients.html, 1999.
11. C. Fournet and A. Schmitt. An implementation of Ambients in JoCAML. Software available from http://join.inria.fr/ambients.html, 1999.
12. A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *FoSSaCS'99*, volume 1578 of *LNCS*, pages 212–226, 1999.
13. F. Le Fessant. The JoCAML system prototype. Software and documentation available from http://pauillac.inria.fr/jocaml, 1998.
14. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00*, pages 352–364. ACM, Jan. 2000.
15. U. Nestmann and B. C. Pierce. Decoding choice encodings. In *CONCUR'96*, volume 1119 of *LNCS*, pages 179–194, Aug. 1996. Revised full version as report ERCIM-10/97-R051, 1997.
16. V. Oostrom. Con uence by decreasing diagrams. *Theoretical Computer Science*, 126:259–280, 1994.
17. J. Parrow and P. Sjödin. Multiway synchronization verified with coupled simulation. In *CONCUR'92*, volume 630 of *LNCS*, pages 518–533, 1992.

Appendix: Two Notions of Mobile Computations

We define our syntax and semantics for the calculus of Mobile Ambients and for the distributed join calculus. We refer to the long version of the paper [10] for an overview of the two calculi.

Operational Semantics for Ambients Our syntax and semantics of the calculus of ambients are given in figures 5 and 6.

This presentation slightly differs from Cardelli and Gordon's [4] on several counts. In spirit, it is actually closer to the harness semantics of [12]. Our structural equivalence is more restrictive; it does not introduce or remove νx binders; it operates only in evaluation contexts. Our operational semantics represents the unfolding of replication as a silent reduction step rather than a structural law. Also, communicated values are just ambient names, rather than both names and chains of capabilities. (Chains of capabilities are fully supported in the Jocaml implementation, but their encoding is heavy.)

Operational Semantics for the Join Calculus Our syntax and semantics for the join calculus are given in figures 7 and 1.

A *path* α is a string of location names a, b, ... *Active locations* are locations not under a `def`; they can be nested. The path of an active sublocation $a[D : P]$ is $\alpha.a$, where α is the path of its enclosing location. A *configuration* is a conjunction of top-level locations such that every location has a unique name, such that the set of paths for all active locations is prefix-closed except for the empty prefix (i.e. active locations form a tree whose nodes are indexed by location names), and such that every channel is defined in at most one location. In a configuration, a location with path $\alpha.a$ is *folded* when it is the only top-level location whose path contains a.

Names can be bound either as parameters $\widetilde{y}$ in a message pattern $x(\widetilde{y})$ or as names defined in D by $\texttt{def}\ D\ \texttt{in}\ P$. The definition $a[D' : P]$ defines a and names defined in D'. A definition containing a rule with message pattern $x(\widetilde{y})$ also defines x.

To simplify the translation of section 2, we supplement the join calculus with some convenient extensions, which are easily encoded in the plain join calculus. We supplement definitions with new constructs $\texttt{uid}\ i$ and $\texttt{fresh}\ a$ that bind names i and a, which we use to generate unique identifiers—we could use instead dummy rules such as $i() \triangleright 0$. We use a record notations $e = \{l_1 = x_1; \ldots l_n = x_n\}$ as a shortcut for a tuple of names $x_1, \ldots, x_n$ passed in a consistent order, and write $e.l_i$ for x_i. We use an algebraic notation In $b\ \kappa$, Out $b\ \kappa$ for log entries with tags In, Out and names b, κ. We use finite sets of log entries, interpreted in the standard mathematical sense. Rather than making explicit a standard encoding of set iterators for implementing the process $Flush(l, in, out, \kappa)$, we supplement the operational semantics of the join calculus with a rule for flushing logs of messages:

$$\textsc{Flush}\quad Flush(l, in, out, \kappa) \to \prod_{\text{In}\ d\ \kappa' \in l\ \mid\ \kappa \neq \kappa'} in(d, \kappa') \quad \mid \quad \prod_{\text{Out}\ d\ \kappa' \in l\ \mid\ \kappa \neq \kappa'} out(d, \kappa')$$

$P ::=$		ambient process
	$n[\,P\,]$	ambient
$\mid$	$P \mid P'$	parallel composition
$\mid$	$C.P$	guarded process
$\mid$	$\nu n.P$	name restriction
$\mid$	$\langle n \rangle$	asynchronous message
$\mid$	$(x).P$	message reception
$\mid$	$!P$	replication
$\mid$	$\mathbf{0}$	inert process
$C ::=$		capability
	$\texttt{in}\ n$	ingoing migration
$\mid$	$\texttt{out}\ n$	outgoing migration
$\mid$	$\texttt{open}\ n$	ambient dissolution

Figure 5. Syntax for the ambient calculus

Evaluation contexts $E(\cdot)$ are defined by the grammar

$$E(\cdot) ::= \ \cdot \ \mid \ P \mid E(\cdot) \ \mid \ E(\cdot) \mid P \ \mid \ n[\,E(\cdot)\,] \ \mid \ \nu n.E(\cdot)$$

Structural equivalence $\equiv$ is the smallest equivalence relation closed by application of evaluation contexts, by α-conversion, and such that

$$\begin{array}{lll} \text{P0} & P \mid \mathbf{0} \equiv P & \text{R1} \ \dfrac{n \text{ is not free in } P}{P \mid \nu n.Q \equiv \nu n.(P \mid Q)} \\ \text{P1} & P \mid P' \equiv P' \mid P & \\ \text{P2} & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') & \text{R2} \ \dfrac{m \neq n}{m[\nu n.P] \equiv \nu n.m[P]} \end{array}$$

Ambient reduction $\rightarrow$ is the smallest relation closed by structural equivalence, by application of evaluation contexts, and such that

$$\textsc{In} \quad \begin{array}{l} m[\,P\,] \mid n[\,\texttt{in}\ m.Q \mid R\,] \\ \rightarrow m\big[\,P \mid n[\,Q \mid R\,]\,\big] \end{array} \qquad \textsc{Out} \quad \begin{array}{l} m\big[\,P \mid n[\,\texttt{out}\ m.Q \mid R\,]\,\big] \\ \rightarrow m[\,P\,] \mid n[\,Q \mid R\,] \end{array}$$

$$\textsc{Open} \quad \texttt{open}\ n.Q \mid n[\,R\,] \rightarrow Q \mid R$$

$$\textsc{Recv} \quad \langle n \rangle \mid (x).P \rightarrow P\{^{n}/_{x}\} \qquad\qquad \textsc{Repl} \quad !P \rightarrow P \mid !P$$

Figure 6. Operational semantics for the ambient calculus

$P ::=$		join calculus process
	$\mathbf{0}$	inert process
$\mid$	$P \mid P'$	parallel composition
$\mid$	$x(\widetilde{y})$	asynchronous message
$\mid$	$\mathtt{go}(a); P$	migration request
$\mid$	$\mathtt{def}\ D\ \mathtt{in}\ P$	local definition
$D ::=$		join calculus definition
	$\top$	void definition
$\mid$	$D \wedge D'$	composition
$\mid$	$J \triangleright P$	reaction rule
$\mid$	$a[D : P]$	sub-location (named a, running D and P)
$\mid$	$\alpha[D : P]$	top-level location (with path α, running D and P)
$J ::=$		join pattern
	$x(\widetilde{y})$	message pattern
$\mid$	$J \mid J'$	synchronization

Figure 7. Syntax for the distributed join calculus

Structural equivalence $\equiv$ (on both processes and definitions) is the smallest equivalence relation closed by application of contexts $\cdot \wedge \cdot$, $\cdot \mid \cdot$ and $\alpha[\ \cdot : \cdot\ ]$, by α-conversion on bound names, and such that:

$$\begin{array}{lll} \text{P0} & P \mid \mathbf{0} \equiv P & \text{D0} \quad D \wedge \top \equiv D \\ \text{P1} & P \mid P' \equiv P' \mid P & \text{D1} \quad D \wedge D' \equiv D' \wedge D \\ \text{P2} & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') & \text{D2} \quad (D \wedge D') \wedge D'' \equiv D \wedge (D' \wedge D'') \end{array}$$

$$\textsc{Tree}\ \begin{array}{c} \alpha\big[a[D' : P'] \wedge D : P\big] \\ \equiv \alpha.a[D' : P'] \wedge \alpha[D : P] \end{array} \qquad \textsc{Scope}\ \frac{\text{names defined in } D' \text{ are fresh}}{\begin{array}{c}\alpha[D : P \mid \mathtt{def}\ D'\ \mathtt{in}\ P'] \\ \equiv \alpha[D \wedge D' : P \mid P']\end{array}}$$

Join calculus reduction $\rightarrow$ is the smallest relation on configurations that is closed by structural equivalence and such that:

$$\textsc{Comm}\ \frac{x \text{ is defined in } D'}{\begin{array}{l}\alpha[D : x(\widetilde{v}) \mid P] \wedge \beta[D' : P'] \wedge E \\ \rightarrow \alpha[D : P] \wedge \beta[D' : x(\widetilde{v}) \mid P'] \wedge E\end{array}} \qquad \textsc{Join}\ \frac{\sigma \text{ operates on message contents of } J}{\begin{array}{l}\alpha[D \wedge J \triangleright Q : J\sigma \mid P] \wedge E \\ \rightarrow \alpha[D \wedge J \triangleright Q : Q\sigma \mid P] \wedge E\end{array}}$$

$$\textsc{Go}\ \frac{a \text{ folded}}{\begin{array}{l}\alpha.a[D : P \mid \mathtt{go}(b); Q] \wedge \beta.b[D' : P'] \wedge E \\ \rightarrow \beta.b.a[D : P \mid Q] \wedge \beta.b[D' : P'] \wedge E\end{array}}$$

Figure 8. Operational semantics for the distributed join calculus

Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness

Naoki Kobayashi

Department of Information Science, University of Tokyo
koba@is.s.u-tokyo.ac.jp

Abstract. In our previous papers [7,11,23], we presented advanced type systems for the π-calculus, which can guarantee deadlock-freedom in the sense that certain communications will eventually succeed *unless the whole process diverges*. Although such guarantee is quite useful for reasoning about the behavior of concurrent programs, there still remains a weakness that the success of a communication cannot be completely guaranteed due to the problem of divergence. For example, while a server process that has received a request message cannot discard the request, it is allowed to infinitely delegate the request to other processes, causing a *livelock*. In this paper, we show that we can guarantee not only deadlock-freedom but also livelock-freedom, by modifying our previous type systems for deadlock-freedom. The resulting type system guarantees that certain communications will eventually succeed under fair scheduling, no matter whether processes diverge. Moreover, it can also guarantee that some of those communications will succeed within a certain amount of time.

1 Introduction

It is an important and challenging task to statically guarantee the correctness of concurrent programs: Because concurrent programs are more complex than sequential programs (due to dynamic control, non-determinism, deadlock, etc.), it is hard for programmers to debug concurrent programs or reason about their behavior. Recent advanced use of the Internet is further increasing the importance of correctness of concurrent programs: It is now becoming common that programs (which may be written by untrusted or malicious programmers) are distributed through the Internet and they run concurrently at multiple sites, interacting with each other.

Unfortunately, existing concurrent/distributed programming languages or thread libraries provide only limited support for the correctness of concurrent programs. Some of them were proposed as extensions of *typed* functional languages [16,21], but their type systems do not give much information about the behavior of a concurrent program: for example, in CML [21], a term of a function type may not even behave like a function: it may get stuck or return non-deterministic results.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 365–389, 2000.
© Springer-Verlag Berlin Heidelberg 2000

To improve the above situation, a number of type systems have been studied through process calculi, just as type systems for functional languages have been studied through λ-calculus. Our type systems [7,11,23] for the π-calculus [13,14] are among the most powerful ones. They can guarantee that every well-typed process is deadlock-free, in the sense that certain communications will eventually succeed unless they diverge. In the most recent type system [11], a programmer just needs to declare which communications they want to succeed, and if the whole process is judged to be well-typed, it is indeed guaranteed that those communications succeed unless the process diverges. For example, in the server-client model, a client process can be written as $(\nu r)\,(s![request, r] \mid r?^{\mathbf{c}}[reply].\,P)$ in the π-calculus-like language of [11]. (νr) creates a fresh communication channel for receiving a reply from the server. $s![request, r]$ sends a request and the channel to the location $\mathtt{s}$ (which is also a channel) of a server. In parallel to this, $r?^{\mathbf{c}}[reply].\,P$ waits for a reply from the server. The annotation $\mathbf{c}$ to ? indicates that a reply should eventually arrive. If the whole system of processes (including the server) is well typed, then a reply is indeed guaranteed to arrive (again, unless the whole system diverges): the server eventually receives the request message, and sends a reply. In this way, the type system can ensure that a process implementing a server really behaves like a server and that a channel implementing a semaphore is really used as a semaphore (a process that has acquired a semaphore will eventually release it). The type systems are reasonably expressive: In the previous paper [7], we have shown that our type system can guarantee deadlock-freedom of the simply-typed λ-calculus with various reduction strategies and typical concurrent objects. Nestmann [15] used our type system to show that his encoding of the choice operator into a choice-free fragment of the π-calculus does not introduce deadlock.

Although the above deadlock-freedom property is quite useful for reasoning about the behavior of concurrent programs, there is still a limitation: the success of a communication is not completely guaranteed because a process may diverge before the communication succeeds. For example, while a server process that has received a request message cannot discard the request, it is allowed to infinitely delegate the request to other processes, causing a *livelock*.

In this paper, we show that we can guarantee livelock-freedom as well as deadlock-freedom, by modifying our previous type systems for deadlock-freedom. The resulting type system guarantees that certain communications will eventually succeed under fair scheduling, *no matter whether processes diverge*. Moreover, it can also guarantee that some of those communications will succeed within a certain amount of time. Surprisingly, this modification to our previous type systems also has a good effect on formalization: Intuitions on types become clearer, and the typing rules become even simpler.

In the rest of this paper, we first introduce our target language and explain what we mean by deadlock and livelock in Section 2. Section 3 reviews basic ideas of our previous type systems for deadlock-freedom. Section 4 introduces our new type system for freedom from livelock and deadlock, and shows its soundness. Section 5 shows that with a minor modification, the type system

can also guarantee that certain communications succeed within a certain time bound. The type system in Section 4 is rather naive and cannot guarantee the livelock-freedom of recursive programs. In Section 6, we show that we can recover the expressive power to some extent by introducing dependent types. Section 7 discusses related work and Section 8 concludes this paper. Due to lack of space, most proofs are omitted: they are found in an accompanying technical report [9]. We do not discuss issues of type check or type reconstruction in this paper. We expect that those issues are basically similar to the case for the deadlock-free calculus [7,11], but complete type reconstruction would be unrealistic for an extension with dependent types.

2 Target Language

This section introduces the target language of our type system and defines deadlock and livelock. The target language is a subset of the polyadic π-calculus [13]. We drop the matching and choice operators from the π-calculus, but keep the other operators. In particular, channels are first-class citizens as in the usual π-calculus, in the sense that they can be dynamically created and passed through other channels. Although this makes it difficult to guarantee deadlock/livelock-freedom, we believe that first-class channels are important in modeling modern concurrent/distributed programming languages. In fact, for example, in concurrent object-oriented programming [25], an object is dynamically created and its reference is passed through messages. Because a reference to a concurrent object corresponds to a record of communication channels [12, 18], channels should be first-class data.

2.1 Syntax

Definition 1 (processes). *The set of processes is defined by the following syntax.*

$$
\begin{array}{lll}
P \ \textit{(processes)} & ::= & \mathbf{0} \mid x!^{a}[v_1,\ldots,v_n].\,P \mid x?^{a}[y_1,\ldots,y_n].\,P \mid (P \mid Q) \mid (\nu x)\,P \\
 & & \mid \textbf{if } v \textbf{ then } P \textbf{ else } Q \mid *P \\
v \ \textit{(values)} & ::= & \textit{true} \mid \textit{false} \mid x \\
a \ \textit{(attributes)} & ::= & \emptyset \mid \mathbf{c}
\end{array}
$$

Here, x and y_is range over a countably infinite set of variables.

Notation 2. As usual, $y_1,\ldots,y_n$ in $x?[y_1,\ldots,y_n].\,P$ and x in $(\nu x)\,P$ are called bound variables. The other variables are called free variables. We assume that α-conversions are implicitly applied so that bound variables are always different from each other and from free variables. $[x_1 \mapsto v_1,\ldots,x_n \mapsto v_n]P$ denotes a process obtained from P by replacing all free occurrences of $x_1,\ldots,x_n$ with $v_1,\ldots,v_n$. We write $\tilde{x}$ for a sequence $x_1,\ldots,x_n$. $[\tilde{x} \mapsto \tilde{v}]$ and $(\nu\tilde{x})$ abbreviate $[x_1 \mapsto v_1,\ldots,x_n \mapsto v_n]$ and $(\nu x_1)\cdots(\nu x_n)$ respectively. We often omit an inaction $\mathbf{0}$ and write $x!^{a}[\tilde{y}]$ for $x!^{a}[\tilde{y}].\,\mathbf{0}$. When attributes are not important,

we omit them and just write $x![\tilde{y}].\,P$ and $x?[\tilde{y}].\,P$ for $x!^a[\tilde{y}].\,P$ and $x?^a[\tilde{y}].\,P$ respectively.

$\mathbf{0}$ denotes inaction. $x!^a[v_1,\ldots,v_n].\,P$ denotes a process that sends a tuple $[v_1,\ldots,v_n]$ on x and then (after the tuple is received by some process) behaves like P. An attribute a expresses the programmer's intention and it does not affect the operational semantics: $a = \mathbf{c}$ means that the programmer wants this output to succeed, i.e., once the output is executed and the tuple is sent, the tuple is expected to be received eventually. There is no such requirement when a is $\emptyset$. $x?^a[y_1,\ldots,y_n].\,P$ denotes a process that receives a tuple $[v_1,\ldots,v_n]$ on x and then behaves like $[y_1 \mapsto v_1,\ldots,y_n \mapsto v_n]P$. If a is $\mathbf{c}$, then this input is expected by the programmer to succeed eventually. $P \mid Q$ denotes a concurrent execution of P and Q, and $(\nu x)\,P$ denotes a process that creates a fresh channel x and then behaves like P. **if** v **then** P **else** Q behaves like P if v is *true* and behaves like Q if v is *false*; otherwise it is blocked forever. $*P$ represents infinitely many copies of the process P running in parallel.

Example 3. A process $*succ?[m,n,r].\,r![m+n]$ behaves as a function server computing the sum of two integers. (For clarity, we extend the language with integers and operations on them here.) It receives a triple consisting of two integers and a channel, and sends the sum of the integers to the channel. A client process can be written like $(\nu y)\,(succ![1,2,y] \mid y?^{\mathbf{c}}[x].\,P)$. The attribute $\mathbf{c}$ of the input process declares that the result should be eventually received.

Example 4. A binary semaphore (or lock) can be implemented by using a channel. Basically, we can regard the presence of a value in the channel as the unlocked state, and the absence of a value as the locked state. Then, creation of a semaphore corresponds to channel creation, followed by output of a null tuple $((\nu x)\,(x![] \mid P))$. The semaphore can be acquired by extracting a value from the channel $(x?[].\,Q)$, and released by putting a value back into the channel $(x![])$. If we want to make sure that the semaphore can be eventually acquired, we can annotate the input as $x?^{\mathbf{c}}[].\,Q$.

2.2 Operational Semantics

The operational semantics is essentially the same as the standard reduction semantics of the π-calculus [13]. For a subtle technical reason, we introduce a structural preorder instead of a structural congruence relation. The only differences from the usual structural congruence $\equiv$ are that $*P \mid P \succeq *P$ does not hold and that $\succeq$ is not closed under output and input prefixes and conditionals.

Definition 5. *The* structural preorder $\succeq$ *is the least reflexive and transitive relation closed under the following rules ($P \equiv Q$ denotes $(P \succeq Q) \wedge (Q \succeq P)$):*

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu x)\,(P \mid Q) \equiv (\nu x)\,P \mid Q (x \text{ not free in } Q)$$

$$*P \succeq *P \mid P \qquad \frac{P \succeq P' \qquad Q \succeq Q'}{P \mid Q \succeq P' \mid Q'}$$

$$\frac{P \succeq Q}{(\nu x)\, P \succeq (\nu x)\, Q} \qquad \frac{P \succeq Q}{*P \succeq *Q}$$

Now we define the reduction relation. Following the operational semantics of the linear π-calculus [10], we define the reduction relation as a ternary relation $P \xrightarrow{l} Q$. l expresses on which channel the reduction is performed: l is either ϵ, which means that the reduction is performed by communication on an internal channel or by the reduction of a conditional expression, or x, which means that the reduction is performed by communication on the free channel x.

Definition 6. *The reduction relation* $\xrightarrow{l}$ *is the least relation closed under the following rules:*

$$x!^{a}[\tilde{v}].\, P \mid x?^{a'}[\tilde{z}].\, Q \xrightarrow{x} P \mid [\tilde{z} \mapsto \tilde{v}]Q$$

$$\frac{P \xrightarrow{l} Q}{P \mid R \xrightarrow{l} Q \mid R} \qquad \frac{P \xrightarrow{x} Q}{(\nu x)\, P \xrightarrow{\epsilon} (\nu x)\, Q}$$

$$\frac{P \xrightarrow{l} Q \qquad l \neq x}{(\nu x)\, P \xrightarrow{l} (\nu x)\, Q} \qquad \frac{P \succeq P' \qquad P' \xrightarrow{l} Q' \qquad Q' \succeq Q}{P \xrightarrow{l} Q}$$

$$\textbf{if } \textit{true} \textbf{ then } P \textbf{ else } Q \xrightarrow{\epsilon} P \qquad \textbf{if } \textit{false} \textbf{ then } P \textbf{ else } Q \xrightarrow{\epsilon} Q$$

Notation 7. We write $P \longrightarrow Q$ if $P \xrightarrow{l} Q$ for some l. We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$. $P \xrightarrow{l}$ and $P \longrightarrow$ mean $\exists Q.(P \xrightarrow{l} Q)$ and $\exists Q.(P \longrightarrow Q)$ respectively.

2.3 Deadlock-Freedom and Livelock-Freedom

Based on the above operational semantics, we formally define deadlock-freedom and livelock-freedom below. Basically, we regard a process as deadlocked or livelocked if one of its subprocesses is trying to communicate with some process but blocked forever without finding a communication partner. However, not every process that is blocked forever should be regarded as being in a bad state. For example, it should be no problem that a server process waits for a request forever: it just means that no client process sends a request message. It is also fine that an output process remains forever on a channel implementing a semaphore (Example 4), because it means that no process tries to acquire the semaphore. Therefore, we focus our attention on communications annotated with the attribute **c**. A process is considered deadlocked or livelocked if it is trying to perform input or output but blocked forever, *and if the input or output is annotated with* **c**.

We first define deadlock.

Definition 8 (deadlock, deadlock-freedom). *A process P is in* deadlock *if (i) $P \succeq (\nu\tilde{y})\,(x?^{\mathbf{c}}[\tilde{z}].\,Q \mid R)$ or $P \succeq (\nu\tilde{y})\,(x!^{\mathbf{c}}[\tilde{z}].\,Q \mid R)$ and (ii) there is no P' such that $P \longrightarrow P'$. A process P is* deadlock-free *if there exists no Q such that $P \longrightarrow^* Q$ and Q is in deadlock.*

Remark 9. In the usual terminology, the deadlock often refers to a more restricted state, where processes are blocked forever because of *cyclic dependencies on communications*. As the above definition shows, in this paper, the deadlock refers to a state where processes are blocked forever, irrespectively of whether or not there are cyclic dependencies. Our definition of deadlock subsumes the usual, narrower definition of deadlock.

Example 10. $(\nu x)\,(x?^{\mathbf{c}}[\,].\,\mathbf{0})$ is in deadlock because the input from x is annotated with $\mathbf{c}$ but there is no output process. $(\nu x)\,(\nu y)\,(x?^{\mathbf{c}}[\,].\,y![\,] \mid y?[\,].\,x![\,])$ is also deadlocked because the input on x cannot succeed because of cyclic dependencies on communications on x and y. On the other hand, $(\nu x)\,(x?[\,].\,\mathbf{0})$ and $(\nu x)\,(x![\,] \mid x?^{\mathbf{c}}[\,].\,\mathbf{0})$ are not in deadlock.

Example 11. A process $(\nu x)\,(x?^{\mathbf{c}}[\,].\,\mathbf{0} \mid (\nu y)\,(y![x] \mid *y?[z].\,y![z]))$ is deadlock-free. Although the input from x never succeeds, the entire process is never blocked.

The last example shows a weakness of the deadlock-freedom property: Even if a process is deadlock-free, it is not completely guaranteed that communications eventually succeed. The livelock-freedom property defined below requires that communications eventually succeed, no matter whether the process diverges.

To give a reasonable definition of livelock, we need to assume that scheduling is *fair*, in the sense that every communication that is enabled infinitely many times eventually succeeds.

Definition 12 (fair reduction sequence). *A reduction sequence $P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \cdots$ is fair if the following conditions hold.*

(i) If there exists an infinite increasing sequence $n_1 < n_2 < \ldots$ of natural numbers such that $P_{n_i} \succeq (\nu\tilde{w}_i)\,(x!^{a}[\tilde{v}].\,Q \mid x?^{a_i}[\tilde{y}].\,Q_i \mid R_i)$, then there exists $n \geq n_1$ such that $P_n \succeq (\nu\tilde{w})\,(x!^{a}[\tilde{v}].\,Q \mid x?^{a'}[\tilde{y}].\,Q' \mid R')$ and $(\nu\tilde{w})\,(Q \mid [\tilde{y} \mapsto \tilde{v}]Q' \mid R') \succeq P_{n+1}$.

(ii) If there exists an infinite increasing sequence $n_1 < n_2 < \ldots$ of natural numbers such that $P_{n_i} \succeq (\nu\tilde{w}_i)\,(x?^{a}[\tilde{y}].\,Q \mid x!^{a_i}[\tilde{v}_i].\,Q_i \mid R_i)$, then there exists $n \geq n_1$ such that $P_n \succeq (\nu\tilde{w})\,(x?^{a}[\tilde{y}].\,Q \mid x!^{a'}[\tilde{v}].\,Q' \mid R')$ and $(\nu\tilde{w})\,([\tilde{y} \mapsto \tilde{v}]Q \mid Q' \mid R') \succeq P_{n+1}$.

(iii) If $P_i \succeq (\nu\tilde{w})\,(\mathbf{if}\ v\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2 \mid R)$ and $v = true$ or $false$ for some i, then either the reduction sequence is finite or there exists $n \geq i$ such that $P_n \succeq (\nu\tilde{w})\,(\mathbf{if}\ v\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2 \mid R')$, $(\nu\tilde{w})\,(Q' \mid R') \succeq P_{n+1}$, and $Q' = Q_1$ if $v = true$ and $Q' = Q_2$ otherwise.

Note that by definition, every finite reduction sequence is fair.

Definition 13 (full reduction sequence). *A reduction sequence* $P_1 \longrightarrow P_2 \longrightarrow \cdots$ *is* full *if it is an infinite reduction sequence or a finite reduction sequence ending with* P_n *such that* $P_n \not\longrightarrow$,

Now we define the livelock-freedom property. Intuitively, a process is livelock-free if in any fair reduction sequence, a process trying to perform communication with capability annotation can eventually communicate with another process.

Definition 14 (livelock-freedom). *A process* P_0 *is livelock-free if the following conditions hold for any full fair reduction sequence* $P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \cdots$:

1. *If* $P_i \succeq (\nu\tilde{w})\,(x!^{\mathbf{c}}[\tilde{v}].\,Q \mid R)$ *for some* $i \geq 0$, *there exists* $n \geq i$ *such that* $P_n \succeq (\nu\tilde{w'})\,(x!^{\mathbf{c}}[\tilde{v}].\,Q \mid x?^{a}[\tilde{y}].\,R_1 \mid R_2)$ *and* $(\nu\tilde{w'})\,(Q \mid [\tilde{y} \mapsto \tilde{v}]R_1 \mid R_2) \succeq P_{n+1}$.
2. *If* $P_i \succeq (\nu\tilde{w})\,(x?^{\mathbf{c}}[\tilde{y}].\,Q \mid R)$ *for some* $i \geq 0$, *there exists* $n \geq i$ *such that* $P_n \succeq (\nu\tilde{w'})\,(x?^{\mathbf{c}}[\tilde{y}].\,Q \mid x!^{a}[\tilde{v}].\,R_1 \mid R_2)$ *and* $(\nu\tilde{w'})\,([\tilde{y} \mapsto \tilde{v}]Q \mid R_1 \mid R_2) \succeq P_{n+1}$.

Again, our definition of livelock may defer from the usual terminology. In our definition, livelock-freedom is a strictly stronger property than deadlock-freedom.

Example 15. The process in Example 11 is deadlock-free but not livelock-free. A process $(\nu x)\,(x![\,] \mid x?^{\mathbf{c}}[\,].\,\mathbf{0}) \mid (\nu y)\,(y![\,] \mid *y?[\,].\,y![\,])$ is livelock-free: Under the fairness assumption, the input from x eventually succeeds.

3 Previous Type Systems for Deadlock-Freedom

The basic idea of our previous type systems for deadlock-freedom [7,11,23] is to extend ordinary channel types [5,24,17] with more precise information on how channels are used. The extra information can be classified into the channel-wise behavior, expressed by using *usages* [11,23], and the inter-channel dependency on the order of communications, expressed by using *time tags* [7]. We first explain usages and time tags, and then sketch the type system.

3.1 Usages

A usage of a channel describes in which order the channel can be used for input and/or output. For example, we denote by 0 the usage of a channel that cannot be used at all, and by $I.O.0$ the usage of a channel that can be first used for input and then used for output. $U_1 \,||\, U_2$ denotes the usage of a channel that can be used according to U_1 by one process and U_2 by another process. $*U$ denotes the usage of a channel that can be used according to U by infinitely many processes. The usage of an input-only channel [17] is expressed as $*I.0$, and that of a linear (use-once) channel [10] is expressed as $I.0 \,||\, O.0$. So, usages are generalization of polarities and multiplicities of the linear π-calculus [10]. A channel used as a binary semaphore (Example 4) has the usage $O.0 \,||\, *I.O.0$.

Capabilities and Obligations. Because we are concerned with whether each input or output succeeds, we associate each I and O with a set of two complementary attributes called *capabilities* (denoted by $\mathbf{c}$) and *obligations* (denoted by $\mathbf{o}$). The capability attribute indicates that the communication is guaranteed to succeed. A guaranteed input ($?^{\mathbf{c}}$) or output operation ($!^{\mathbf{c}}$) in Section 2 is allowed only when I or O is annotated with this input capability. We can refine the usage of a binary semaphore given above as $O.0 \,||\, *I_{\mathbf{c}}.O.0$, to indicate that the semaphore can be eventually acquired. In order for the semaphore to be always acquired, however, the output denoted by O *must* be executed. It is expressed by the obligation attribute. So, the correct usage of a binary semaphore is $O_{\mathbf{o}}.0 \,||\, *I_{\mathbf{c}}.O_{\mathbf{o}}.0$. Similarly, the usage of a linear channel is refined as $O_{\mathbf{co}}.0 \,||\, I_{\mathbf{co}}.0$, meaning that the channel must be used exactly once for input and output, and that the communication always succeeds. The usage of a channel used for client-server connection can be denoted by $*I_{\mathbf{o}}.0 \,||\, *O_{\mathbf{c}}.0$. $*I_{\mathbf{o}}.0$ means that a server must accept infinitely many request messages, and $*O_{\mathbf{c}}.0$ means that clients can successfully send infinitely many request messages.

Channel-Wise Consistency. Creating a channel of some bad usage results in deadlock. For example, creating a channel of usage $I_{\mathbf{c}}.0 \,||\, O.0$ may cause deadlock, because an input must succeed, but an output operation may not be executed. So, we require that the usage of each channel must be consistent (called *reliable*) in the sense that each input/output capability is guaranteed by the corresponding output/input obligation. This condition excludes out a deadlocked process $(\nu x)\, x!^{\mathbf{c}}[\,]$, which uses x according to the inconsistent usage $O_{\mathbf{c}}.0$.

3.2 Time Tags

Controlling the channel-wise behavior is not sufficient to guarantee deadlock-freedom. For example, consider the process $x?^{\mathbf{c}}[\,].\, y![\,] \,|\, y?^{\mathbf{c}}[\,].\, x![\,]$. It uses channels x and y according to a consistent usage $I_{\mathbf{c}}.0 \,|\, O_{\mathbf{o}}.0$ but it is in deadlock. This is because of the following cyclic dependencies between x and y: The condition that the input from x is a capability depends on the condition that the output on x is executed (i.e., is an obligation), which depends on the condition that the input from y succeeds (i.e., is a capability). But it further depends on the output obligation on y, which depends on the input capability on x.

To avoid such cyclic dependencies, we associate each channel with a time tag and maintain the order of time tags. Let t_x be the time tag of a channel x and t_y be that of y. Then, we allow a process to use capabilities on x before fulfilling obligations on y only if there is an ordering $t_x < t_y$. In the above example, $x?^{\mathbf{c}}[\,].\, y![\,]$ is allowed under the ordering $t_x < t_y$ but the other subprocess $y!^{\mathbf{c}}[\,].\, x![\,]$ is disallowed because it tries to use an input capability on y before fulfilling an output obligation on x.

3.3 Typing

By using usages and time tags, we can express the type of a channel in the form $[\tau_1, \ldots, \tau_n]^t/U$. The part $[\tau_1, \ldots, \tau_n]$ means that the channel is used for passing

a tuple of values of types $\tau_1, \ldots, \tau_n$ ($\tau_1, \ldots, \tau_n$ may also be channel types). U is the usage of the channel, and t is the time tag.

A type judgment is of the form $x_1 : \tau_1, \ldots, x_n : \tau_n; \mathcal{R} \vdash P$, where $\mathcal{R}$ is a strict partial order on time tags. It means that P uses $x_1, \ldots, x_n$ according to types $\tau_1, \ldots, \tau_n$, and obeys the order $\mathcal{R}$ in performing communications. We give examples of type judgments.

- $x : []^{t_x}/{*I_{\mathbf{c}}.O_{\mathbf{o}}.0}, y : []^{t_y}/{*I_{\mathbf{c}}.O_{\mathbf{o}}.0}; \{(t_x, t_y)\} \vdash P$: The usage ${*I_{\mathbf{c}}.O_{\mathbf{o}}.0}$ of x and y means that P uses x and y as binary semaphores. $\{(t_x, t_y)\}$ indicates that P acquires x first when it acquires both semaphores at the same time: If it acquired x (i.e., used the input capability on x) first, it would obtain the obligation to release x (i.e., the output obligation on x), so it could not use the capability to acquire the semaphore y before fulfilling the obligation to release x.
- $f : [[int]^{t_r}/O_{\mathbf{o}}.0]^{t_f}/{*I_{\mathbf{o}}.0}; \emptyset \vdash P$: The usage of f means that P tries to receive infinitely many messages from f. The type $[int]^{t_r}/O_{\mathbf{o}}.0$ means that a received value is a channel for which an integer must be sent. So, the above judgment indicates that P behaves like a server providing an integer: P waits to receive a channel repeatedly, and each time it receives a channel, it sends an integer back to the received channel.

The following typing rule for input best illustrates how usages and time tags are enforced by our type system [11]:

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]^{t_x}/U, y_1 : \tau_1, \ldots, y_n : \tau_n; \mathcal{R} \vdash P \quad (hasob(\Gamma) \vee a = \mathbf{c}) \Rightarrow (b = \mathbf{c} \vee b = \mathbf{co}) \quad t_x \mathcal{R} \Gamma}{\Gamma, x : [\tau_1, \ldots, \tau_n]^{t_x}/I_b.U; \mathcal{R} \vdash x?^a[y_1, \ldots, y_n].\, P}$$

Because the first condition of the premise implies that P uses x according to U and because $x?^a[y_1, \ldots, y_n].\, P$ uses x for input before that, the total usage of x is of the form $I_b.U$. The condition $hasob(\Gamma)$ means that Γ contains an obligation on some channel, i.e., that an input or output operation must be performed on some channel. If so, the input from x must succeed. The attribute b should therefore contain the capability attribute. The last condition $t_x \mathcal{R} \Gamma$ means that if Γ contains an obligation on a channel, t_x should be less than the time tag of that channel.

The following rule for channel creation makes sure that only channels of consistent usages are created.

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]^{t_x}/U; \mathcal{R} \vdash P \quad reliable(U)}{\Gamma; \mathcal{R} \vdash (\nu x)\, P}$$

In this manner, it is enforced that each channel is consistently used and that there is no cyclic dependency on communications through different channels. Thus, every closed well-typed process is guaranteed to be deadlock-free (see [7, 11] for formal proofs).

4 Type System for Livelock-Freedom

This section introduces our new type system that can guarantee the livelock-freedom property. We first analyze weaknesses of our previous type systems and explain how to remove them (in Section 4.1). Then we define our new type system and show that it is sound in the sense that every closed well-typed process is livelock-free.

4.1 Basic Ideas

As mentioned in Section 1, a weakness of the type systems for deadlock-freedom [7,11,23] is that it cannot completely guarantee the success of a communication because of the problem of divergence. In fact, the process $(\nu x)\,(x?^{\mathbf{c}}[\,].\,\mathbf{0} \mid (\nu y)\,(y![x] \mid *y?[z].\,y![z]))$ in Example 11 is well typed [11, 23]: We can assign to x a type $[\,]^{t_x}/(I_{\mathbf{c}}.0 \,||\, O_{\mathbf{o}}.0)$ and to y a type $[[\,]^{t_x}/O_{\mathbf{o}}.0]^{t_y}/(O_{\mathbf{c}}.\mathbf{0} \,||\, *I_{\mathbf{o}}.O_{\mathbf{c}}.\mathbf{0})$. The type of y demands that any process that has received a channel from y must send a value to it, or delegate an obligation to do so to other processes (by forwarding the received channel). The above process $*y?[z].\,y![z]$ certainly obeys this requirement: it forwards the received channel z to y, and according to the type of y, a receiver of z is supposed to fulfill the obligation to send a value to z. However, the receiver is actually this process itself and therefore the obligation is never fulfilled.

The above problem comes from the fact that a received obligation x of type $[\,]^{t_x}/O_{\mathbf{o}}.0$ can be just forwarded as the same obligation of type $[\,]^{t_x}/O_{\mathbf{o}}.0$. Actually, however, because it takes some time to forward x, it should forward x as the obligation that should be fulfilled within a *shorter* time limit. This observation leads us to replace the binary information on whether or not an obligation exists with a time limit within which an obligation must be fulfilled. Absence of an obligation is the special case where the time limit is infinite. Similarly, the capability attribute is also replaced by the maximum amount of time required before the communication succeeds. Thus, a usage $O_a.0$ is replaced by $O^{t_o}_{t_c}.0$, which means that an output operation must be executed within time t_o (by an output or input operation being executed, we mean that a process becomes ready to output or input a value, not that a process succeeds to find the communication partner and complete the communication), and if the output is executed, it is guaranteed to succeed within time t_c.

Now, let us reconsider the process $(\nu x)\,(x?^{\mathbf{c}}[\,].\,\mathbf{0} \mid (\nu y)\,(y![x] \mid *y?[z].\,y![z]))$. Suppose that the channel y carries a channel for which an output must be executed within time t_o. Then, when the process $*y?[z].\,y![z]$ receives x, some time has already been spent for the communication on y. So, the process must fulfill the obligation to output on x within the time t_o *minus* the time spent for the communication on y. It therefore cannot forward x through y, which may require another time of length t_o until the output is performed. In this manner, infinite delegations of an obligation is forbidden, and livelock-freedom is guaranteed as a result.

The above change has also a good effect on formalization. An unpleasant point about the previous type systems was that the meaning of a channel type is not completely clear by itself because of time tags: The meaning of the time tags in a channel type is only determined by the order relative to time tags attached to other channel types. Now, the time tags are no longer required (actually they have been integrated into time bounds of capabilities and obligations). For example, the dependency between x and y in $x?[\,].\,y![\,]$ is captured by the condition "The input from x should succeed within a time limit shorter than the time limit of the fulfillment of an obligation to output on y." This condition can be expressed in the typing rules more neatly than the corresponding condition on time tags in the previous type systems [11].

4.2 Time Quantum

Now we define our type system. We first introduce *time quantums*, which are used for giving time bounds of capabilities and obligations. We could define a time quantum simply as a natural number, expressing the number of reduction steps. To keep the flexibility, however, we use the following slightly more complicated definition.

Definition 16 (time quantum). *The set of time quantums, ranged over by t, is defined by:*

$$t ::= 0 \mid \mathbf{t} \mid t_1 + t_2 \mid n \cdot t$$

Here, $\mathbf{t}$ ranges over the carrier set $\mathbf{T}$ of a well-founded order $\mathcal{T} = (\mathbf{T}, \ll)$. An element of $\mathbf{T}$ is called a time unit*. We assume that $\mathbf{T}$ contains the least time unit $\mathbf{t_{min}}$ and the greatest time unit $\mathbf{t}_\infty$. n ranges over the set* **Nat** *of natural numbers.*

Intuitively, $\mathbf{t}_1 \ll \mathbf{t}_2$ means that $\mathbf{t}_1$ is sufficiently shorter than $\mathbf{t}_2$ so that the summation of any finite number of $\mathbf{t}_1$s is shorter than $\mathbf{t}_2$. $\mathbf{t_{min}}$ represents the time required for one step reduction of a process. $\mathbf{t}_\infty$ represents an infinite length of time. These intuitions are expressed in the following definitions of the semantics of a time quantum and the order between time quantums.

Definition 17 (semantics of time quantum). *Let t be a time quantum. A mapping $[\![t]\!]$ from $\mathbf{T}$ to* **Nat** *is defined by:*

$$[\![0]\!](\mathbf{t}) = 0 \qquad [\![\mathbf{t}]\!](\mathbf{t}') = \begin{cases} 1 \text{ if } \mathbf{t} = \mathbf{t}' \\ 0 \text{ otherwise} \end{cases}$$

$$[\![t_1 + t_2]\!](\mathbf{t}) = [\![t_1]\!](\mathbf{t}) + [\![t_2]\!](\mathbf{t}) \qquad [\![n \cdot t]\!](\mathbf{t}) = n \times [\![t]\!]\mathbf{t}$$

Remark 18. We do not distinguish between time quantums whose semantics are the same. For example, we identify $(\mathbf{t}_1 + \mathbf{t}_2) + (\mathbf{t}_2 + \mathbf{t}_3)$ with $\mathbf{t}_1 + 2 \cdot \mathbf{t}_2 + \mathbf{t}_3$. With this identification, every time tag can be expressed in a normal form $\Sigma_{\mathbf{t}_i \in \mathbf{T}} n_i \cdot \mathbf{t}_i$.

Definition 19. *The binary relation $\leq$ on time quantums is defined by: $t_1 \leq t_2$ if and only if either (i) $[\![t_2]\!](\mathbf{t}_\infty) > 0$, or (ii) for each $\mathbf{t} \in \mathbf{T}$, either $[\![t_1]\!](\mathbf{t}) \leq [\![t_2]\!](\mathbf{t})$ or there exists $\mathbf{t}'$ such that $[\![t_1]\!](\mathbf{t}') < [\![t_2]\!](\mathbf{t}')$ and $\mathbf{t} \ll \mathbf{t}'$. We write $t_1 < t_2$ if $(t_1 \leq t_2) \wedge \neg(t_2 \leq t_1)$*

It is trivial that the binary relation $\leq$ on time tags is a preorder.

Example 20. Suppose $\mathbf{t}_1 \ll \mathbf{t}_2 \ll \mathbf{t}_\infty$. Then, $n \cdot \mathbf{t}_1 < \mathbf{t}_2$ for any $n \in \mathbf{Nat}$. $\mathbf{t}_\infty + t_1 \leq \mathbf{t}_\infty + t_2$ holds for any time quantums t_1 and t_2. So, $\mathbf{t}_\infty + t$ is essentially equivalent to $\mathbf{t}_\infty$.

The following lemma states an important property to guarantee livelock-freedom. It follows from the fact that the set of time units is well-founded.

Lemma 21. *The set of time quantums is well-founded with respect to $<$, i.e., there is no infinite decreasing sequence $t_0 > t_1 > t_2 > \cdots$.*

4.3 Usages and Types

As mentioned in Section 4.1, we replace the capability/obligation attributes of a usage with two time quantums t_o, t_c and remove the time tag from a channel type.

Definition 22 (usages). *The set $\mathcal{U}$ of usages is given by the following syntax.*

$$U ::= 0 \mid \alpha^{t_o}_{t_c}.U \mid (U_1 \,||\, U_2) \mid *U$$
$$\alpha ::= I \mid O$$

Notation 23. We give a higher precedence to prefixes ($\alpha^{t_o}_{t_c}.$ and $*$) than to $||$. So, $I^{t_o}_{t_c}.U_1 \,||\, U_2$ means $(I^{t_o}_{t_c}.U_1) \,||\, U_2$, not $I^{t_o}_{t_c}.(U_1 \,||\, U_2)$. We sometimes write $\overline{\alpha}$ for I or O. It denotes O when $\alpha = I$ and I when $\alpha = O$.

$O^{t_o}_{t_c}.U$ denotes the usage of a channel that can be first used for output, and then used according to U. Intuitively, t_o means that an output process must be executed within time t_o. If $\mathbf{t}_\infty \leq t_o$, the output need not be performed. t_c means that the output is guaranteed to succeed within time t_c after it is executed. More precisely, t_c is the time limit within which a communication partner is found and the communication starts: It takes $\mathbf{t_{min}}$ more until the communication is completed. If $\mathbf{t}_\infty \leq t_c$, then the output is not guaranteed to succeed. $I^{t_o}_{t_c}.U$ denotes the usage of a channel that must be first used for input within time t_o, and then used according to U if the input succeeds. The input is guaranteed to succeed within time t_c. The meaning of the other usage constructors is the same as that in the previous type systems (see Section 3).

Definition 24 (types). *The set of types is given by the following syntax.*

$$\tau ::= \mathit{bool} \mid [\tau_1, \ldots, \tau_n]/U$$

$[\tau_1, \ldots, \tau_n]/U$ denotes the type of a channel that can be used for communicating a tuple of values of types $\tau_1, \ldots, \tau_n$. The channel must be used according to the usage U.

We introduce several operations on usages and types. $\boxed{t}U$ represents the usage of a channel that is used according to U *after a delay of at most* t.

Definition 25. *A unary operation* $\boxed{t}$ *on usages is defined inductively by:*

$$\boxed{t}0 = 0 \qquad \boxed{t}\alpha^{t_o}_{t_c}.U = \alpha^{t_o+t}_{t_c}.U$$
$$\boxed{t}(U_1 \,||\, U_2) = \boxed{t}U_1 \,||\, \boxed{t}U_2 \qquad \boxed{t}(*U) = *\boxed{t}U$$

Constructors and operations on usages are extended to operations on types.

Definition 26. *Operations* $||, *, \boxed{t}$ *on types are defined by:*

$$bool \,||\, bool = bool \qquad ([\tilde{\tau}]/U_1) \,||\, ([\tilde{\tau}]/U_2) = [\tilde{\tau}]/(U_1 \,||\, U_2)$$
$$*bool = bool \qquad *[\tilde{\tau}]/U_1 = [\tilde{\tau}]/*U_1$$
$$\boxed{t}bool = bool \qquad \boxed{t}[\tilde{\tau}]/U_1 = [\tilde{\tau}]/\boxed{t}U_1$$

4.4 Reliability of Usages

As in the type systems for deadlock-freedom (see Section 3), the usage of each channel must be consistent (reliable) in the sense that each input/output capability is guaranteed by an output/input obligation. For example, consider a usage $I^{\mathbf{t}_\infty}_{t_c}.U_1 \,||\, O^{t_o}_{\mathbf{t}_\infty}.U_2$. In order for the success of input to be guaranteed, it must be the case that $t_o \leq t_c$. This consistency should be preserved during the whole computation. After communication on a channel of usage $I^{\mathbf{t}_\infty}_{t_c}.U_1 \,||\, O^{t_o}_{\mathbf{t}_\infty}.U_2$ happens, the channel is used according to $U_1 \,||\, U_2$. So, $U_1 \,||\, U_2$ must also be consistent. To state such a condition, we first define *reduction* of a usage.

Definition 27. $\cong$ *is the least congruence relation satisfying the following rules:*

$$U \cong U \,||\, 0 \qquad *0 \cong 0$$
$$U_1 \,||\, U_2 \cong U_2 \,||\, U_1 \qquad (U_1 \,||\, U_2) \,||\, U_3 \cong U_1 \,||\, (U_2 \,||\, U_3)$$
$$*U \cong *U \,||\, U \qquad *(U_1 \,||\, U_2) \cong *U_1 \,||\, *U_2$$
$$**U \cong *U$$

Definition 28 (usage reduction). *A binary relation* $\longrightarrow$ *on usages is the least relation closed under the following rules: (i)*$I^{t_o}_{t_c}.U_1 \,||\, O^{t'_o}_{t'_c}.U_2 \,||\, U_3 \longrightarrow U_1 \,||\, U_2 \,||\, U_3$*, and (ii)*$U_1 \longrightarrow U_2$ *if* $U_1 \cong U'_1$*,* $U'_1 \longrightarrow U'_2$*, and* $U'_2 \cong U_2$*.*

Now, a usage U is defined to be reliable if after any reduction steps, whenever it contains an input/output capability (i.e., it is structurally congruent to $I^{t_o}_{t_c}.U_1 \,||\, U_2$), it contains an output/input obligation of an equal or shorter time bound. We first define a predicate to judge whether a usage contains an obligation, and then give a formal definition of the reliability.

Definition 29 (obligation). *The relation* $ob_I, ob_O (\subseteq \mathcal{U})$ *between a time quantum and a usage are defined by:* $ob_\alpha(t, U)$ *if and only if* $\mathbf{t}_\infty \leq t$ *or* $\exists t_o, t_c, U_1, U_2.((U \cong \alpha^{t_o}_{t_c}.U_1 \,||\, U_2) \wedge (t_o \leq t))$.

Definition 30 (reliability). *U is* reliable*, written $rel(U)$, if $ob_\alpha(t_c, U_2)$ holds whenever $U \longrightarrow^* \cong \overline{\alpha}^{t_o}_{t_c}.U_1 \,||\, U_2$. A type τ is reliable, written $rel(\tau)$, if it is of the form $[\tilde{\tau}]/U$ and $rel(U)$ holds.*

Remark 31. The above definitions of ob_α and the reliability do not completely match the intuition on usages. Consider a usage $U = O^{\mathbf{t}}_{\mathbf{t}_\infty}.\mathbf{0} \,||\, *I^{\mathbf{t}_\infty}_{\mathbf{t}}.O^{\mathbf{t}}_{\mathbf{t}_\infty}.\mathbf{0}$ of a binary semaphore. According to the above definitions, U is reliable. Actually, however, an input operation may not succeed within time $\mathbf{t}$: Although an output operation is always executed within $\mathbf{t}$, the input may be delayed because of other input operations. If another process succeeds to input from the channel, it must wait for another period of length $\mathbf{t}$. So, U should not be reliable. (Under the fair scheduling, every input succeeds after a finite number of output operations. So, according to the intuition, $O^{\mathbf{t}}_{\mathbf{t}_\infty}.0 \,||\, *I^{\mathbf{t}_\infty}_{\mathbf{t}'}.O^{\mathbf{t}}_{\mathbf{t}_\infty}.0$ should be reliable if $\mathbf{t} \ll \mathbf{t}'$.) For the guarantee of livelock-freedom, however, the above definition of the reliability is sufficient and much simpler. We will redefine the reliability in Section 5 to guarantee time-boundedness.

4.5 Subusage and Subtyping

The subusage relation defined below allows one usage to be viewed as another usage. For example, consider the usage $I^{\mathbf{t}_\infty}_{t_c}.0 \,||\, I^{\mathbf{t}_\infty}_{t_c}.0$ of a channel that can be used twice for input, possibly in parallel. Because the input is not an obligation, it should be allowed to use the channel for input just once for now, and use once more only after the input succeeds, as expressed by $I^{\mathbf{t}_\infty}_{t_c}.I^{\mathbf{t}_\infty}_{t_c}.0$. Such a relation between usages is expressed by $I^{\mathbf{t}_\infty}_{t_c}.0 \,||\, I^{\mathbf{t}_\infty}_{t_c}.0 \le I^{\mathbf{t}_\infty}_{t_c}.I^{\mathbf{t}_\infty}_{t_c}.0$.

Definition 32. *The* subusage *relation $\le$ on usages is the least reflexive and transitive relation closed under the following rules:*

$$\frac{U \cong U'}{U \le U'} \quad (\textsc{SubU-Cong}) \qquad \frac{\mathbf{t}_\infty \le t_o}{\alpha^{t_o}_{t_c}.U \le 0} \quad (\textsc{SubU-Zero})$$

$$\frac{U_1 \le U_1' \qquad U_2 \le U_2'}{U_1 \,||\, U_2 \le U_1' \,||\, U_2'} \quad (\textsc{SubU-Par}) \qquad \frac{U \le U'}{*U \le *U'} \quad (\textsc{SubU-Rep})$$

$$\alpha^{t_o}_{t_c}.U_1 \,||\, \boxed{t_o + t_c + \mathbf{t_{min}}} U_2 \le \alpha^{t_o}_{t_c}.(U_1 \,||\, U_2) \quad (\textsc{SubU-Delay})$$

$$\frac{U \le U' \qquad t_o' \le t_o \qquad t_c \le t_c'}{\alpha^{t_o}_{t_c}.U \le \alpha^{t_o'}_{t_c'}.U'} \quad (\textsc{SubU-Act})$$

The rule (SubU-Zero) indicates that a channel with no obligation need not be used at all. The rule (SubU-Delay) allows some usage to be delayed until a communication succeeds. In the usage $\alpha^{t_o}_{t_c}.(U_1 \,||\, U_2)$, an obligation contained in U_2 is delayed until the operation α (which is I or O) is executed and it succeeds. Because the time t_o may pass before it is executed and another time period of length t_c may pass before it is enabled, the time $t_o + t_c + \mathbf{t_{min}}$ may be required before obligations in U_2 is fulfilled. It is taken into account by the operation

$\boxed{t_o + t_c + \mathbf{t_{min}}}$ in the lefthand side. The rule (SUBU-ACT) means that it is safe to estimate the time bound of an obligation to be shorter than the actual time bound, while the time bound of a capability should be estimated to be longer than the actual bound.

The subusage relation can be extended to the following subtyping relation.

Definition 33 (subtyping). *A subtyping relation $\leq$ is the least relation closed under the rules: (i)$bool \leq bool$, and (ii) $[\tilde{\tau}]/U \leq [\tilde{\tau}]/U'$ if $U \leq U'$.*

Definition 34. *$noob(\tau)$ if $\tau = bool$ or $\tau = [\tilde{\tau}]/U$ and $U \leq 0$.*

4.6 Type Environment

A type environment is a mapping from a finite set of variables to types. We use a metavariable Γ for a type environment. We use a sequence $v_1 : \tau_1, \ldots, v_n : \tau_n$ to denote the type environment Γ such that $dom(\Gamma) = \{v_1, \ldots, v_n\} \backslash \{true, false\}$ and $\Gamma(v_i) = \tau_i$ for each $v_i \in dom(\Gamma)$. In the sequence, if $v_i = true$ or $v_i = false$, τ_i must be $bool$. (So, the sequence $x : \tau, true : bool$ denotes the same type environment as $x : \tau$. The sequence $true : [\tau]/U$ is invalid.) We write $\emptyset$ for the type environment whose domain is empty. When $x \notin dom(\Gamma)$, we write $\Gamma, x : \tau$ for the type environment Γ' satisfying $dom(\Gamma') = dom(\Gamma) \cup \{x\}$, $\Gamma'(x) = \tau$, and $\Gamma'(y) = \Gamma(y)$ for $y \in dom(\Gamma)$. $\Gamma \backslash \{x_1, \ldots, x_n\}$ denotes the type environment Γ' such that $dom(\Gamma') = dom(\Gamma) \backslash \{x_1, \ldots, x_n\}$ and $\Gamma'(x) = \Gamma(x)$ for each $x \in dom(\Gamma')$.

The operations on types are pointwise extended to those on type environments as follows.

Definition 35 (operations on type environments). *The operations $||, *, \boxed{t}$ on types are defined by:*

$$dom(\Gamma_1 \,||\, \Gamma_2) = dom(\Gamma_1) \cup dom(\Gamma_2)$$

$$(\Gamma_1 \,||\, \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \,||\, \Gamma_2(x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in dom(\Gamma_1) \backslash dom(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_2) \backslash dom(\Gamma_1) \end{cases}$$

$$dom(*\Gamma) = dom(\Gamma) \qquad (*\Gamma)(x) = *(\Gamma(x))$$
$$dom(\boxed{t}\Gamma) = dom(\Gamma) \qquad (\boxed{t}\Gamma)(x) = \boxed{t}(\Gamma(x))$$

Definition 36. *Γ is reliable, written $rel(\Gamma)$, if $rel(\Gamma(x))$ holds for each $x \in dom(\Gamma)$.*

Definition 37. *A binary relation $\leq$ on type environments is defined by: $\Gamma_1 \leq \Gamma_2$ if and only if (i) $dom(\Gamma_1) \supseteq dom(\Gamma_2)$, (ii) $\Gamma_1(x) \leq \Gamma_2(x)$ for each $x \in dom(\Gamma_2)$, and (iii) $noob(\Gamma_1(x))$ for each $x \in dom(\Gamma_1) \backslash dom(\Gamma_2)$.*

$$\emptyset \vdash \mathbf{0} \quad \text{(T-Zero)} \qquad \frac{\Gamma, x:[\tau_1,\ldots,\tau_n]/U \vdash P \qquad rel(U)}{\Gamma \vdash (\nu x)\,P} \quad \text{(T-New)}$$

$$\frac{\Gamma_1 \vdash P_1 \qquad \Gamma_2 \vdash P_2}{\Gamma_1 \,||\, \Gamma_2 \vdash P_1 \mid P_2} \quad \text{(T-Par)} \qquad \frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\boxed{\mathbf{t_{min}}}\,\Gamma \,||\, v:bool \vdash \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q} \quad \text{(T-If)}$$

$$\frac{\Gamma \vdash P}{*\Gamma \vdash *P} \quad \text{(T-Rep)} \qquad \frac{\Gamma' \vdash P \qquad \Gamma \le \Gamma'}{\Gamma \vdash P} \quad \text{(T-Weak)}$$

$$\frac{\Gamma, x:[\tau_1,\ldots,\tau_n]/U \vdash P \qquad a = \mathbf{c} \Rightarrow t_c < \mathbf{t}_\infty}{\boxed{t_c + \mathbf{t_{min}}}(v_1:\tau_1 \,||\, \cdots \,||\, v_n:\tau_n \,||\, \Gamma) \,||\, x:[\tau_1,\ldots,\tau_n]/O^{t_o}_{t_c}.U \vdash x!^a[v_1,\ldots,v_n].\,P} \quad \text{(T-Out)}$$

$$\frac{\Gamma, x:[\tau_1,\ldots,\tau_n]/U, y_1:\tau_1,\ldots,y_n:\tau_n \vdash P \qquad a = \mathbf{c} \Rightarrow t_c < \mathbf{t}_\infty}{\boxed{t_c + \mathbf{t_{min}}}\,\Gamma, x:[\tau_1,\ldots,\tau_n]/I^{t_o}_{t_c}.U \vdash x?^a[y_1,\ldots,y_n].\,P} \quad \text{(T-In)}$$

Fig. 1. Typing Rules

4.7 Typing Rules

The typing rules are shown in Figure 1. Each rule is explained below.

(T-Par) The premises imply that P_1 uses variables as described by Γ_1, and in parallel to this, P_2 uses variables as described by Γ_2. So, the type environment of $P_1 \mid P_2$ should be represented as the combination $\Gamma_1 \,||\, \Gamma_2$. For example, if $\Gamma_1 = x:[\,]/I^{t_o}_{t_c}.0$ and $\Gamma_2 = x:[\,]/O^{t'_o}_{t'_c}.0$, then $P_1 \mid P_2$ should be well typed under $x:[\,]/(I^{t_o}_{t_c}.0 \,||\, O^{t'_o}_{t'_c}.0)$.

(T-Rep) Because $*P$ runs infinitely many copies of P in parallel and the premise implies that each P uses variables according to Γ, $*P$ uses variables according to $*\Gamma$ as a whole.

(T-New) $(\nu x)\,P$ is well typed if P is well typed and it uses x as a channel of a reliable usage.

(T-If) Since **if** v **then** P **else** Q executes either P or Q, P and Q must be well typed under the same type environment Γ. Assuming that it takes the time $\mathbf{t_{min}}$ to check whether v is *true* or *false*, we can express the total use of variables by **if** v **then** P **else** Q as $\boxed{\mathbf{t_{min}}}\,\Gamma \,||\, v:bool$.

(T-Weak) $\Gamma \le \Gamma'$ means that Γ represents a more liberal use of variables than Γ'. So, if P is well typed under Γ', so is under Γ.

(T-Out) The lefthand premise implies that P uses x as a channel of usage U and uses other variables according to Γ. Because $x!^a[v_1,\ldots,v_n].\,P$ uses x for output before that, the total usage of x is expressed by $O^{t_o}_{t_c}.U$. Other variables are used by P and the receiver of $[v_1,\ldots,v_n]$ according to $v_1:\tau_1 \,||\, \cdots \,||\, v_n:\tau_n \,||\, \Gamma$ only after the communication on x succeeds. Because it may take the time t_c until the communication is enabled and it takes $\mathbf{t_{min}}$ more before the

communication completes, the total use of other variables is estimated as $\boxed{t_c + \mathbf{t_{min}}}(v_1 : \tau_1 \,||\, \cdots \,||\, v_n : \tau_n \,||\, \Gamma)$. The righthand premise requires that the time bound of the capability must be finite if the input is annotated with **c**.

(T-In) This is similar to the rule (T-Out). Because P uses x according to the usage U, the total usage of x is expressed as $I^{t_o}_{t_c}.U$. Because $x?^a[y_1, \ldots, y_n].P$ executes P only after the communication on x has completed, the total use of other variables is estimated as $\boxed{t_c + \mathbf{t_{min}}}\Gamma$.

4.8 Type Soundness

Now we show that our type system is sound in the sense that any closed well-typed process is livelock-free (Corollary 42). We omit most of the proofs but present a proof sketch of the main theorem (Theorem 41) because it may be particularly interesting: While the usual type soundness refers to a safety property that a bad thing never happens, livelock-freedom is a liveness property that a good thing eventually happens. Full proofs are given in the full version of this paper [9].

Type soundness comes from the subject reduction theorem (Theorem 39), which states that well-typedness of a process is preserved by reduction, *plus* a property that some progress is always made by reduction (Lemma 40). As in the linear π-calculus, the type environment of a process may change during reduction. For example, while a process $x![\,] \,|\, x?[\,].\mathbf{0}$ is well typed under $x : [\,]/(O^{t_o}_{t_c}.0 \,||\, I^{t'_o}_{t'_c}.0)$, the reduced process $\mathbf{0}$ is well typed under $x : [\,]/0$. This change of a type environment is captured by the following relation $\Gamma \xrightarrow{l} \Delta$.

Definition 38. *A 3-place relation $\Gamma \xrightarrow{l} \Delta$ is defined to hold if one of the following conditions holds:*

1. *$l = \epsilon$ and $\Gamma = \Delta$.*
2. *$l = x$, $\Gamma = (\Gamma', x : [\tilde{\tau}]/U)$, $\Delta = (\Gamma', x : [\tilde{\tau}]/U')$, and $U \longrightarrow U'$ for some $\Gamma', \tilde{\tau}, U$, and U'.*

We write $\Gamma \longrightarrow \Delta$ when $\Gamma \xrightarrow{l} \Delta$ for some l.

Theorem 39 (subject reduction). *If $\Gamma \vdash P$ and $P \xrightarrow{l} Q$, then there exists Δ such that $\Delta \vdash Q$ and $\Gamma \xrightarrow{l} \Delta$.*

The following lemma states that after a communication succeeds, the processes that have been blocked by the communication can fulfill obligations within shorter time bounds.

Lemma 40. *If $\Gamma, x : [\tilde{\tau}]/U \vdash x!^a[\tilde{v}].P \,|\, x?^{a'}[\tilde{y}].Q$, then there exist Δ, U' such that $\Delta, x : [\tilde{\tau}]/U' \vdash P \,|\, [\tilde{y} \mapsto \tilde{v}]Q$, $\Gamma \leq \boxed{\mathbf{t_{min}}}\Delta$, and $U \longrightarrow U'$.*

The following main theorem says that any obligation with a finite time bound is eventually fulfilled (the properties A and B below), and that any capability with a finite time bound can eventually be used (the properties C and D).

Theorem 41. *Suppose that $P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \cdots$ is a full fair reduction sequence. If $t < \mathbf{t}_\infty$, the following properties hold.*

- *A (Output Obligation). If (i) $\Gamma \vdash P_0$, (ii) $rel(\Gamma)$, (iii) $\Gamma = \Gamma', x:[\tilde{\tau}]/U$, and (iv) $ob_O(t, U)$, then there exists $n \geq 0$ such that $P_n \succeq (\nu\tilde{w})\,(x!^a[\tilde{v}].\,Q_1 \mid Q_2)$.*
- *B (Input Obligation). If (i) $\Gamma \vdash P_0$, (ii) $rel(\Gamma)$, (iii) $\Gamma = \Gamma', x:[\tilde{\tau}]/U$, and (iv) $ob_I(t, U)$, then there exists $n \geq 0$ such that $P_n \succeq (\nu\tilde{w})\,(x?^a[\tilde{y}].\,Q_1 \mid Q_2)$.*
- *C (Output Capability). If (i) $P_0 \succeq x!^a[\tilde{v}].\,Q \mid R$, (ii) $\Delta_1, x:[\tilde{\tau}]/O_t^{t_o}.U_1 \vdash x!^a[\tilde{v}].\,Q$, (iii) $\Delta_2 \vdash R$, and (iv) $rel((\Delta_1, x:[\tilde{\tau}]/O_t^{t_o}.U_1) \,||\, \Delta_2)$, then there exists $n \geq 0$ such that $P_n \succeq (\nu\tilde{w})\,(x!^a[\tilde{v}].\,Q \mid x?^{a'}[\tilde{y}].\,R_1 \mid R_2)$ and $(\nu\tilde{w})\,(Q \mid [\tilde{y} \mapsto \tilde{v}]R_1 \mid R_2) \succeq P_{n+1}$.*
- *D (Input Capability). If (i) $P_0 \succeq x?^a[\tilde{y}].\,Q \mid R$, (ii) $\Delta_1, x:[\tilde{\tau}]/I_t^{t_o}.U_1 \vdash x?^a[\tilde{y}].\,Q$, (iii) $\Delta_2 \vdash R$, and (iv) $rel((\Delta_1, x:[\tilde{\tau}]/I_t^{t_o}.U_1) \,||\, \Delta_2)$, then there exists $n \geq 0$ such that $P_n \succeq (\nu\tilde{w})\,(x?^a[\tilde{y}].\,Q \mid x!^{a'}[\tilde{v}].\,R_1 \mid R_2)$ and $(\nu\tilde{w})\,([\tilde{y} \mapsto \tilde{v}]Q \mid R_1 \mid R_2) \succeq P_{n+1}$.*

Proof Sketch. The proof proceeds by induction on t (notice that the relation $<$ on the subset $\{t \mid t < \mathbf{t}_\infty\}$ of time quantums is well-founded: there is no infinite decreasing sequence). We show only the properties A and C. B and D are similar.

Suppose the theorem holds for any t' such that $t' < t$.

A. By the assumption $\Gamma \vdash P_0$, there must exist P_{01} and P_{02} such that (a) $P_0 \succeq (\nu\tilde{w})\,(P_{01} \mid P_{02})$, (b) $\Gamma_1, x:[\tilde{\tau}]/(O_{t_c}^t.U_3 \,||\, U_4) \vdash P_{01}$, and (c) P_{01} is an output, input, or conditional process. Without loss of generality, we can assume that $(\nu\tilde{w})\,(P_{01} \mid P_{02}) \longrightarrow P_1$. If P_{01} is of the form $x!^a[\tilde{v}].\,R$, the result immediately holds. Otherwise, P_{01} must be **if** b **then** P'_{01} **else** P''_{01} with b being *true* or *false*, or trying to use an input or output capability on another channel with a time limit t'_c shorter than t.

In the former case, by the assumption of the fairness, P_{01} is eventually reduced to P'_{01} or P''_{01}. By the typing rules, P'_{01} and P''_{01} are well typed under $\Gamma_1, x:[\tilde{\tau}]/(O_{t''_c}^{t'}.U_3 \,||\, U_4)$ for some t' such that $t' + \mathbf{t_{min}} \leq t$ (which implies $t' < t$ as $t < \mathbf{t}_\infty$). So, the required result follows from the property A of induction hypothesis.

In the latter case, by the property C of induction hypothesis, P_{01} must be eventually reduced. By Theorem 39 and Lemma 40, the resulting process is well typed under some type environment $\Delta, x:[\tilde{\tau}]/(O_{t''_c}^{t'}.U_5 \,||\, U_6)$ such that $t' < t$. (See the full paper [9] for details.) So, the required result follows from the property A of induction hypothesis.

C. Suppose there exists no such n. Then, it must be the case that $P_i \succeq x!^a[\tilde{v}].\,Q \mid R_i$ and $R \longrightarrow R_1 \longrightarrow R_2 \longrightarrow \cdots$ is a full fair reduction sequence. We show that there exist infinitely many i such that $R_i \succeq (\nu\tilde{w})\,(x?^{a'}[\tilde{y}].\,R'_i \mid R''_i)$, which contradicts with the assumption that the reduction sequence $P_0 \longrightarrow P_1 \longrightarrow P_2 \longrightarrow \cdots$ is fair. Let j be an arbitrary natural number. Then, by subject reduction (Theorem 39), $\Delta', x:[\tilde{\tau}]/U' \vdash R_j$ and $\Delta_2 \longrightarrow^* (\Delta', x:[\tilde{\tau}]/U')$ for some Δ' and U'. By the assumption (iv) and the definition of the reliability, $ob_I(t, U')$ must hold. We also

have that $\Gamma', x:[\tilde{\tau}]/(O^{t_o}_t.U_1 \,||\, U') \vdash x!^a[\tilde{v}].Q \,|\, R_j$ and $rel(\Gamma')$ for $\Gamma' = \Delta_1 \,||\, \Delta'$. Therefore, by the property B, there exists $i \geq j$ such that $R_i \succeq (\nu\tilde{w})\,(x?^{a'}[\tilde{y}].R'_i \,|\, R''_i)$. Thus, we have shown that there exist infinitely many i such that $R_i \succeq (\nu\tilde{w})\,(x?^{a'}[\tilde{y}].R'_i \,|\, R''_i)$. □

Corollary 42 (livelock freedom). *If $\emptyset \vdash P$, then P is livelock-free.*

Proof. Suppose $\emptyset \vdash P$ and $P \longrightarrow^* P_i \succeq (\nu\tilde{w})\,(x!^{\mathbf{c}}[\tilde{v}].Q \,|\, R)$. By Theorem 39, $\tilde{w}:\tilde{\tau} \vdash x!^{\mathbf{c}}[\tilde{v}].Q \,|\, R$ and $rel(\tilde{w}:\tilde{\tau})$. Moreover, by the typing rules, it must be the case that $\Delta_1, x:[\tilde{\tau'}]/O^{t_o}_{t_c}.U_1 \vdash x!^{\mathbf{c}}[\tilde{v}].Q$, $t_c < \mathbf{t}_\infty$, $\Delta_2 \vdash R$, and $rel((\Delta_1, x:[\tilde{\tau'}]/O^{t_o}_{t_c}.U_1) \,||\, \Delta_2)$. So, the required result follows from the property C of Theorem 41. The case for input is similar. □

5 Time-Bounded Processes

In this section, we briefly show that with a minor modification, our type system can guarantee not only that certain communications *eventually* succeed, but also that some of them succeed within a certain number of (parallel) reduction steps.

5.1 Time-Boundedness

We first define the time-boundedness of a process. We extend the syntax of processes to allow a programmer to declare an upper-bound of the number of reduction steps required until an input or output operation succeeds.

$$P ::= \cdots \mid x!^n[\tilde{v}].P \mid x?^n[\tilde{y}].P$$

The annotation n of $x!^n[\tilde{v}].P$ indicates that the programmer wants this output to succeed within n *parallel* reduction steps (defined below) after it is executed. (Strictly speaking, the output process can *find* its communication partner within n reduction steps and complete the communication in the next step.)

In counting the number of reduction steps, we assume unlimited parallelism, so that communications on different channels can occur in parallel. To model such parallel reduction, we introduce parallel reduction relations $P \Longrightarrow Q$ and $P \overset{S}{\Longrightarrow} Q$ where S is a set of channels. $P \Longrightarrow Q$ means that P is reduced to Q by reducing every conditional expression and performing one communication on every channel whenever possible. $P \overset{S}{\Longrightarrow} Q$ is the same as $P \Longrightarrow Q$ except that communications on free channels occur only on those in the set S. We assume here that at most one communication can occur on each channel and that the two processes to communicate is chosen randomly. So, it takes two steps to reduce $*x?[].\mathbf{0} \,|\, x![] \,|\, x![]$ to $*x?[].\mathbf{0}$. It would be possible to change reduction rules and the type system so as to make as many communications as possible occur simultaneously on each channel and/or to reflect a certain scheduling.

Definition 43 (parallel process reduction). $\stackrel{S}{\Longrightarrow}$ *(where S is a set of variables) and* $\Longrightarrow$ *are the least relations closed under the rules given below.* a^- *is defined by:* $n^- = n-1$, $\mathbf{c}^- = \mathbf{c}$, *and* $\emptyset^- = \emptyset$.

$$x!^a[\tilde{v}].\,P \mid x?^{a'}[\tilde{z}].\,Q \stackrel{\{x\}}{\Longrightarrow} P \mid [\tilde{z} \mapsto \tilde{v}]Q \qquad \mathbf{0} \stackrel{\emptyset}{\Longrightarrow} \mathbf{0}$$

$$x!^a[\tilde{v}].\,P \stackrel{\emptyset}{\Longrightarrow} x!^{a^-}[\tilde{v}].\,P \qquad x?^a[\tilde{v}].\,P \stackrel{\emptyset}{\Longrightarrow} x?^{a^-}[\tilde{v}].\,P$$

$$\textbf{if } \mathit{true} \textbf{ then } P \textbf{ else } Q \stackrel{\emptyset}{\Longrightarrow} P \qquad \textbf{if } \mathit{false} \textbf{ then } P \textbf{ else } Q \stackrel{\emptyset}{\Longrightarrow} Q$$

$$\frac{P \stackrel{S_1}{\Longrightarrow} P' \qquad Q \stackrel{S_2}{\Longrightarrow} Q' \qquad S_1 \cap S_2 = \emptyset}{P \mid Q \stackrel{S_1 \cup S_2}{\Longrightarrow} P' \mid Q'} \qquad \frac{P \stackrel{\emptyset}{\Longrightarrow} Q}{*P \stackrel{\emptyset}{\Longrightarrow} *Q}$$

$$\frac{P \stackrel{S}{\Longrightarrow} Q \qquad x \in S \text{ if } P \stackrel{x}{\longrightarrow}}{(\nu x)\,P \stackrel{S \setminus \{x\}}{\Longrightarrow} (\nu x)\,Q} \qquad \frac{P \succeq P' \qquad P' \stackrel{S}{\Longrightarrow} Q' \qquad Q' \succeq Q}{P \stackrel{S}{\Longrightarrow} Q}$$

$$\frac{P \stackrel{S}{\Longrightarrow} Q \qquad \forall x.(x \in S \text{ if } P \stackrel{x}{\longrightarrow})}{P \Longrightarrow Q}$$

The rules in the first two lines ensure that in each parallel reduction step, every input or output process is either reduced or its time bound is decreased by 1. The rules in the third line ensure that every conditional process is reduced. The righthand premises of the rule for $(\nu x)\,P$ and the last rule make sure that a communication happens on every channel whenever possible.

A process is time-bounded if whenever the time bound of an input or output process has become 0 (i.e., it becomes $x!^0[\tilde{v}].\,P$ or $x?^0[\tilde{y}].\,P$), the input or output operation always succeeds in the next parallel reduction step:

Definition 44 (time-boundedness). *A process P is* time-bounded *if the following conditions hold whenever* $P \Longrightarrow^* P'$.

1. *If* $P' \succeq (\nu\tilde{w})\,(x!^0[\tilde{v}].\,Q \mid R)$, *then* $R \not\stackrel{x}{\longrightarrow}$ *and* $R \succeq (\nu\tilde{u})\,(x?^a[\tilde{y}].\,R_1 \mid R_2)$ *for some* $\tilde{u}, R_1, R_2$.
2. *If* $P' \succeq (\nu\tilde{w})\,(x?^0[\tilde{y}].\,Q \mid R)$, *then* $R \not\stackrel{x}{\longrightarrow}$ *and* $R \succeq (\nu\tilde{u})\,(x!^a[\tilde{v}].\,R_1 \mid R_2)$ *for some* $\tilde{u}, \tilde{v}, R_1, R_2$.

5.2 Modification to the Type System

Now we show how to modify the type system to guarantee the time-boundedness. Because the typing rules in Section 4 already take into account the delay caused by communications on other channels, we just need to refine the reliability condition to estimate the channel-wise behavior more correctly. As stated in Remark 31, a problem of Definition 30 is that it does not take race conditions into account. For example, $O^0_{\mathbf{t}_\infty}.0 \,||\, I_0^{\mathbf{t}_\infty}.O^0_{\mathbf{t}_\infty}.0 \,||\, I_0^{\mathbf{t}_\infty}.O^0_{\mathbf{t}_\infty}.0$ is reliable according to Definition 30, but only one input is guaranteed to succeed immediately: The

other input must wait for a time period of length $\mathbf{t_{min}}$ until an output is executed again. So, the correct usage should be $O^0_{\mathbf{t}_\infty}.0 \,||\, I^{\mathbf{t}_\infty}_{\mathbf{t_{min}}}.O^0_{\mathbf{t}_\infty}.0 \,||\, I^{\mathbf{t}_\infty}_{\mathbf{t_{min}}}.O^0_{\mathbf{t}_\infty}.0$.

We redefine usage reduction to take race conditions into account. For example, $O^0_{\mathbf{t}_\infty}.0 \,||\, I^{\mathbf{t}_\infty}_{\mathbf{t_{min}}}.O^0_{\mathbf{t}_\infty}.0 \,||\, I^{\mathbf{t}_\infty}_{\mathbf{t_{min}}}.O^0_{\mathbf{t}_\infty}.0$ is reduced to $O^0_{\mathbf{t}_\infty}.0 \,||\, I^{\mathbf{t}_\infty}_0.O^0_{\mathbf{t}_\infty}.0$, which can be further reduced to $O^0_{\mathbf{t}_\infty}.0$. To define a new usage reduction relation, we introduce auxiliary relations $\overset{S}{\Longrightarrow}$ where $\{\mathbf{com}, I, O\} \subseteq S$. When $\mathbf{com} \in S$, $U \overset{S}{\Longrightarrow} V$ means that one pair of an input usage and an output usage is reduced. $I \in S$ ($O \in S$, resp.) indicates that an input (output, resp.) action is ready but kept waiting (either because no output action is ready or because another input is chosen for communication).

Definition 45 (timed usage reduction). *Binary relations $\overset{S}{\Longrightarrow}$ and $\Longrightarrow$ ($\{\mathbf{com}, I, O\} \subseteq S$) on usages are the least relations closed under the following rules:*

$$\frac{}{I^{t_o}_{t_c}.U_1 \,||\, O^{t'_o}_{t'_c}.U_2 \overset{\{\mathbf{com}\}}{\Longrightarrow} U_1 \,||\, U_2} \qquad \frac{}{0 \overset{\emptyset}{\Longrightarrow} 0}$$

$$\frac{0 < t_c}{\alpha^{t_o}_{t_c}.U \overset{\{\alpha\}}{\Longrightarrow} \alpha^0_{t_c{}^-}.U} \qquad \frac{0 < t_o}{\alpha^{t_o}_{t_c}.U \overset{\emptyset}{\Longrightarrow} \alpha^{t_o{}^-}_{t_c}.U}$$

$$\frac{U_1 \overset{S_1}{\Longrightarrow} U'_1 \qquad U_2 \overset{S_2}{\Longrightarrow} U'_2 \qquad \mathbf{com} \notin S_1 \cap S_2}{U_1 \,||\, U_2 \overset{S_1 \cup S_2}{\Longrightarrow} U'_1 \,||\, U'_2} \qquad \frac{U \overset{S}{\Longrightarrow} U' \qquad \mathbf{com} \notin S}{*U \overset{S}{\Longrightarrow} *U'}$$

$$\frac{U \cong U' \qquad U' \overset{S}{\Longrightarrow} V' \qquad V' \cong V}{U \overset{S}{\Longrightarrow} V} \qquad \frac{U \overset{S}{\Longrightarrow} V \qquad \mathbf{com} \in S \text{ if } \{I, O\} \subseteq S}{U \Longrightarrow V}$$

Here, $t^- = (n-1) \cdot \mathbf{t_{min}}$ if $[\![t]\!] = [\![n \cdot \mathbf{t_{min}}]\!]$, and $t^- = t$ otherwise.

The left rule in the second line is for the case where the (input or output) action α has already been executed but does not succeed in this reduction step: in this case, the time limit of the capability is reduced by $\mathbf{t_{min}}$. The right rule in the second line is for the case where the action α has not been executed yet: in this case, the time limit of the obligation is reduced by $\mathbf{t_{min}}$. The right rule at the bottom ensures that in each reduction step, if both input and output actions are ready, some pair of an input usage and an output usage must be reduced.

Using the above parallel usage reduction, we can strengthen the reliability condition as defined below. The first condition ensures that when the time limit of an input or output capability has reached 0, the input or output operation must succeed in the next step.

Definition 46 (reliability (refined)). *A usage U is reliable if:*

1. *If $U \Longrightarrow^* \cong \alpha^{t_o}_0.U_1 \,||\, U_2$, then $U_2 \not\longrightarrow$ and there exist t_c, U_3, U_4 such that $U_2 \cong \overline{\alpha}^0_{t_c}.U_3 \,||\, U_4$; and*
2. *$U \longrightarrow^* \cong \alpha^{t_o}_{t_c}.U_1 \,||\, U_2$ implies $ob_{\overline{\alpha}}(t_c, U_2)$.*

We introduce the following typing rules for new processes.

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]/U \vdash P \qquad t_c \leq n \cdot \mathbf{t_{min}}}{\boxed{t_c + \mathbf{t_{min}}}(v_1 : \tau_1 \,||\, \cdots \,||\, v_n : \tau_n \,||\, \Gamma) \,||\, x : [\tau_1, \ldots, \tau_n]/O^{t_o}_{t_c}.U \vdash x!^n[v_1, \ldots, v_n].\, P} \quad \text{(T-BOut)}$$

$$\frac{\Gamma, x : [\tau_1, \ldots, \tau_n]/U, y_1 : \tau_1, \ldots, y_n : \tau_n \vdash P \qquad t_c \leq n \cdot \mathbf{t_{min}}}{\boxed{t_c + \mathbf{t_{min}}}\Gamma, x : [\tau_1, \ldots, \tau_n]/I^{t_o}_{t_c}.U \vdash x?^n[y_1, \ldots, y_n].\, P} \quad \text{(T-BIn)}$$

We can prove the following time-boundedness theorem. The proof is basically similar to that of the livelock-freedom property (Corollary 42): It suffices to prove a parallel version of the subject reduction property (Theorem 39) (although it is harder).

Theorem 47. *Every closed well-typed process is time-bounded.*

6 Extensions with Dependent Types

The typed livelock-free process calculus in Section 4 is not expressive enough. For example, consider the following process P, which is a function server computing the factorial of a natural number:

$$*fact?[n, r].\, (\textbf{if } n = 0 \textbf{ then } r![1] \textbf{ else } (\nu r')\, (fact![n - 1, r'] \,|\, r'?[m].\, r![m \times n])\ )$$

$P \,|\, (\nu y)\, (fact![3, y].\ |\, y?^{\mathbf{c}}[x].\, \mathbf{0})$ cannot be judged to be livelock-free in our type system. Suppose r has type $[\mathbf{Nat}]/O^{t_o}_{t_c}.0$. Because r' is passed through the channel *fact*, r' must have the same type $[\mathbf{Nat}]/O^{t_o}_{t_c}.0$. According to the rule (T-In), in order for $r'?[m].\, r![m \times n]$ to be well typed, it must be the case that $t_o + \mathbf{t_{min}} \leq t_o$, which implies $\mathbf{t}_\infty \leq t_o$. So, it is not guaranteed that P returns a result eventually.

The problem of the above example is that the time bound of an output obligation on r depends on the other argument n. We can use dependent types to express such dependency: In the above process, the type of the channel *fact* can be expressed as $[\Sigma n : \mathbf{Nat}.[\mathbf{Nat}]/O^{2n \cdot \mathbf{t_{min}}}_0.0]/U$.

7 Related Work

Our Previous Type Systems for Deadlock-Freedom. As mentioned earlier, the type system in this paper originates from our previous type systems for deadlock-freedom [7,23,11]. We expect that we can reconstruct a type system for deadlock-freedom by changing the definition of the order between time tags (for example, by identifying $\mathbf{t}_1 + \mathbf{t}_2$ with $\mathbf{t}_2$ if $\mathbf{t}_1 \ll \mathbf{t}_2$). Nice points about the new type system are that the intuitive meaning of a channel type is clearer, and that we can get rid of complex side conditions on time tags (which were introduced to get enough expressive power) in the typing rules of the previous type systems [7,23]. We expect that we can recover much of the expressive power by using more standard concepts like dependent types and polymorphism as described in Section 6.

Other Type Systems for Analyzing Similar Properties of Concurrent Processes. There exist a few other type systems for π-calculus-like languages (where channels are first-class data) that guarantee a livelock-freedom property. As far as we know, however, they deal with more specific situations: Sangiorgi's type system for receptiveness [22] enforces that an input process is spawned immediately after a certain channel (called a receptive name) is created, and therefore guarantees that every output on that channel succeeds immediately. Puntigam and Peter's typed concurrent object calculus [20] guarantees that certain reply messages (which they call promised messages) eventually arrive.

There are a few other type systems that deal with deadlock-freedom [1,19, 26]. Please refer to our previous paper [11] for comparisons with them.

Abadi and Flanagan [4] recently proposed a typed concurrent object calculus that can prevent race conditions. Our type system can also be used to prevent race conditions. Notice that a race condition occurs only when more than one processes try to input or output on the same channel simultaneously. Such a situation can be detected by looking at the usage of each channel: If the usage of a channel can be reduced to a usage of the form $I^{t_o}_{t_c}.U_1 \,||\, I^{t'_o}_{t'_c}.U_2 \,||\, \cdots$, more than one processes may try to perform an input on the channel at the same time. Abadi and Flanagan's race detection [4] is, however, more sophisticated because dependencies between different channels (or locks) are taken into account. We might be able to extend our type system to subsume it by extending a channel usage so that it expresses the use of a group of channels, instead of that of each channel (see discussions on an extension to a deadlock-free concurrent object calculus in [11]).

Type Systems for Bounding Execution Time of Sequential Programs. There are several pierces of work that try to statically bound running-time of sequential programs [6,3]. Most closely related (especially to the extension sketched in Section 6) seems to be Crary and Weirich's work [3], which also uses dependent types. A difficulty in bounding running-time of a concurrent process is that unlike sequential programs where a function/procedure call is never blocked, a process may be blocked until a communication partner becomes ready. We have dealt with this difficulty in this paper by associating each input/output operation with two time bounds: a time bound within which the operation is executed, and another time bound within which a communication partner becomes ready.

Abstract Interpretation. An alternative way to analyze the behavior of a concurrent program would be to use abstract interpretation [2]. Actually, from a very general viewpoint, our type-based analysis of deadlock and livelock can be seen as a kind of abstract interpretation. We can read a type judgment $\Gamma \vdash P$ as "Γ is an abstraction of a concrete process P." (The relation "$\vdash$" corresponds to a pair of abstraction/concretization functions.) Indeed, we can regard a type environment as an abstract process: we have defined reductions of type environments in Section 4.8.

The subject reduction property (Theorem 39) can be interpreted as "whenever a concrete process P is reduced to another concrete process Q, an abstraction

Γ of P can also be reduced to another abstract process Δ which is an abstraction of Q." In other words, every reduction step of a concrete process is simulated by reduction of its abstract process. So, it corresponds to a consistency condition of abstract interpretation. The reliability condition (Definition 30) guarantees that an abstract process never falls into a livelock. Thus, a concrete process is also guaranteed to be livelock-free.

8 Conclusion

In this paper, we have extended our previous type systems for deadlock-freedom to guarantee livelock-freedom and time-boundedness properties. A number of practical issues remain to be solved in applying these type systems to real programming languages, such as how to combine dependent types, polymorphism, etc. to obtain a reasonable expressive power and how and to what extent to let programmers declare type information.

A key idea common to those type systems is to decompose the behavior of a whole process into that on each communication channel, which is specified by using a mini-process calculus of usages. This idea would be applicable to other analyses, such as race detection (as mentioned in Section 7) and memory management. As for an application to memory management, we have already applied a similar idea to analyze how and in which order each heap cell (instead of a communication channel) is used in functional programs [8].

Acknowledgment

We would like to thank Atsushi Igarashi, Eijiro Sumii, and Nobuko Yoshida for useful comments.

References

1. G. Boudol. Typing the use of resources in a concurrent calculus. In *Proceedings of ASIAN'97*, pages 239–253, 1997.
2. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.
3. K. Crary and S. Weirich. Resource bound certification. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 184–198, 2000.
4. C. Flanagan and M. Abadi. Object types against races. In *CONCUR'99*, LNCS 1664, pages 288–303. Springer-Verlag, 1999.
5. S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 429–438, 1993.
6. M. Hofmann. Linear types and non-size-increasing polynomial time computation. In *Proceedings of IEEE Symposium on Logic in Computer Science*, pages 464–473, 1999.

7. N. Kobayashi. A partially deadlock-free typed process calculus. *ACM Transactions on Programming Languages and Systems*, 20(2):436–482, 1998. A preliminary summary appeared in Proceedings of LICS'97, pages 128–139.
8. N. Kobayashi. Quasi-linear types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 29–42, 1999.
9. N. Kobayashi. A livelock-free typed process calculus. Technical report, Dept. Info. Sci., Univ. of Tokyo, 2000. To appear. Available at http://www.yl.is.s.u-tokyo.ac.jp/~koba/publications.html.
10. N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999. Preliminary summary appeared in Proceedings of POPL'96, pp.358-371.
11. N. Kobayashi, S. Saito, and E. Sumii. An implicitly-typed deadlock-free process calculus. Technical Report TR00-01, Dept. Info. Sci., Univ. of Tokyo, January 2000. Available at http://www.yl.is.s.u-tokyo.ac.jp/~koba/publications.html. A summary will appear in Proceedings of CONCUR 2000, LNCS, Springer-Verlag.
12. N. Kobayashi and A. Yonezawa. Towards foundations for concurrent object-oriented programming – types and language design. *Theory and Practice of Object Systems*, 1(4):243–268, 1995.
13. R. Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, 1993.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I, II. *Information and Computation*, 100:1–77, September 1992.
15. U. Nestmann. What is a 'good' encoding of guarded choice? In *EXPRESS'97*, volume 7 of *ENTCS*. Elsevier Science Publishers, September 1997.
16. S. Peyton Jones. Concurrent Haskell. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 295–308, 1996.
17. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
18. B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994)*, LNCS 907, pages 187–215. Springer-Verlag, 1995.
19. F. Puntigam. Coordination requirements expressed in types for active objects. In *Proceedings of ECOOP'97*, LNCS 1241, pages 367–388, 1997.
20. F. Puntigam and C. Peter. Changeable interfaces and promised messages for concurrent components. In *Proceedings of the 1999 ACM Symposium on Applied Computing*, pages 141–145, 1999.
21. J. H. Reppy. CML: A higher-order concurrent language. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, pages 293–305, 1991.
22. D. Sangiorgi. The name discipline of uniform receptiveness (extended abstract). In *Proceedings of ICALP'97*, LNCS 1256, pages 303–313, 1997.
23. E. Sumii and N. Kobayashi. A generalized deadlock-free process calculus. In *Proc. of Workshop on High-Level Concurrent Language (HLCL'98)*, volume 16(3) of *ENTCS*, pages 55–77, 1998.
24. V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In *CONCUR'93*, LNCS 715, pages 524–538. Springer-Verlag, 1993.
25. A. Yonezawa and M. Tokoro. *Object-Oriented Concurrent Programming*. The MIT Press, 1987.
26. N. Yoshida. Graph types for monadic mobile processes. In *FST/TCS'16*, LNCS 1180, pages 371–387. Springer-Verlag, 1996.

Local π-Calculus at Work: Mobile Objects as Mobile Processes*

Massimo Merro[1]**, Josva Kleist[2], and Uwe Nestmann[2]

[1] INRIA, Sophia-Antipolis, France
[2] BRICS - Basic Research in Computer Science,
Danish National Research Foundation,
Aalborg University, Denmark

Abstract. Obliq is a distributed, object-based programming language. In Obliq, the *migration* of an object is proposed as creating a clone of the object at the target site, whereafter the original object is turned into an alias for the clone. Obliq has an only informal semantics, so there is no proof that this style of migration is correct, i.e., transparent to object clients. In this paper, we focus on Øjeblik, an abstraction of Obliq. We give a *π-calculus* semantics to Øjeblik, and we use it to formally prove the correctness of object *surrogation*, an abstraction of object migration.

1 Introduction

The work presented in this paper is in line with the research activity to use the π-calculus as a toolbox for reasoning about distributed object-oriented programming languages. Former works on the semantics of objects as processes have shown the value of this approach: while [22,9,19,10] have focused on just providing formal semantics to object-oriented languages and language features, the work of others [18,20] has been driven by a specific programming problem. Our work tackles a problem in Obliq, Cardelli's lexically-scoped distributed programming language [4]. In this setting, Cardelli proposed to implement object migration by creating a *clone* of the object at the target site and then turning the original (local) object into an *alias* for the new (remote) object. The question arises, whether this style of object migration is correct, and how that can be stated formally. However, Obliq is not equipped with a formal semantics, apart from an unpublished note by Talcott [21], which provides a configuration-style semantics for a subset of Obliq excluding migration. The aim of our project [15] is to remedy this lack of formality and to reason formally about migration.

Previous work Since Obliq is *lexically scoped*, we may ignore the aspects of distribution, at least when regarding the results of Obliq computations, unless distribution sites fail. Following this idea, Øjeblik, which we introduce in Section 3, is an object-based language that represents Obliq's concurrent core [16],

* A draft full paper is available at `http://www.cs.auc.dk/research/FS/ojeblik/`

** Supported by Marie Curie fellowhip, EU-TMR, No. ERBFMBICT983504.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 390–408, 2000.
© Springer-Verlag Berlin Heidelberg 2000

but it can also be seen as a concurrent extension of the Imperative Object Calculus [1]. Øjeblik supports *surrogation*, a distribution-free abstraction of migration.

In [16] we gave a formal definition of *correctness* for object surrogation in Øjeblik which can be straightforwardly extended to object migration in Obliq. The intuition is that, in order to be correct, the surrogation (resp. migration) of an object must be transparent to the clients of that object, i.e., the object must behave the same before and after surrogation (resp. migration). We have formalized this concept as the simple equation $\underline{a.\mathsf{ping} \doteq a.\mathsf{surrogate}}$ where the left side represents the object a before surrogation ($a.\mathsf{ping}$ returns a if reachable), the right side represents the object a after surrogation, and $\doteq$ is an appropriate contextual equivalence, based on the possibility of convergence.

In [16] we have also given several proposals of configuration-style semantics for Øjeblik. One of them fits the Obliq implementation [3,4], but does not guarantee the correctness of object surrogation as defined above. This has been formally shown by exhibiting Øjeblik contexts that are able to distinguish the terms $a.\mathsf{ping}$ and $a.\mathsf{surrogate}$. Similar counterexamples apply to object surrogation in Obliq, as we have tested using the Obliq interpreter [3]. In order to achieve the correctness of surrogation, we have proposed an improved semantics in [16], but that work did not contain a proof.

Contribution of the paper In this paper, we present a π-calculus semantics for Øjeblik corresponding to the aforementioned variant proposed in [16]. We also give a notion of contextual equivalence for objects defined in terms of may convergence on π-processes corresponding to the equivalence $\doteq$. More precisely, our semantics uses *Local* π [12], in short Lπ, a variant of the asynchronous π-calculus [7], where the recipients of a channel are local to the process that has created the channel. We prove the correctness of surrogation in two parts.

The *algebraic* part (Theorem 1) relates, with respect to arbitrary π-calculus contexts, the core component of the translation of a single object in its ping'ed and surrogate'd version—both *after commitment* to the respective request under the condition that the request did *not* arise *internally* from the object itself. Here, we use powerful adaptations of proof techniques, partially known from standard π-calculus and Lπ [12], also to exhibit that the alias-component of the surrogate'd version behaves like a forwarder for external requests (Lemma 6). Due to the unavoidable complexity of the language and its semantics, the proof of Theorem 1 is non-trivial, but it provides the seeked insight that we gave the proper π-calculus semantics to aliased objects—which actually drove the development of the proper corresponding operational semantics to aliased objects in [16].

The *iterative* part (Theorem 2, as conjectured in [16]) relates the may-convergence behavior of the terms $a.\mathsf{ping}$ and $a.\mathsf{surrogate}$, within Øjeblik-contexts. Here, we constructively simulate arbitrarily long converging sequences "up to" Theorem 1, so the Øjeblik-contexts must guarantee that the requests will be external. The main difficulty of Theorem 2 is that inherently concurrent Øjeblik-contexts may non-deterministically prevent either term from eventually committing to

the externally requested operation; this also rules out both the must-variant of convergence equivalence as well as bisimulation equivalences.

Summing up, we give (to our knowledge) the first formal proof that migration can be correctly implemented in terms of cloning and aliasing. Due to lack of space, proofs are sketched or omitted; complete proofs are found in the full paper.

2 Local π: An "Object-Oriented" π-Calculus

Local π [12], in short Lπ, is a variant of the asynchronous π-calculus [7] where, similar to the Join-calculus [5], the recipients of a channel are local to the process that has created the channel. This is achieved by imposing the syntactic constraint that only the output capability of names may be transmitted, i.e., the recipient of a name may only use it in output actions. This property makes Lπ particularly suitable for giving the semantics to, and reasoning about, concurrent object-oriented languages. In particular, we can easily guarantee the uniqueness of object identities—a fundamental feature of objects: in object-oriented languages, the name of an object may be transmitted; the recipient may use that name to access the methods of the object, but it cannot create a new object with the same name. When representing objects in the π-calculus, this translates directly into the constraint that the process receiving an object name may only use it in output actions—a guarantee in our setting.

2.1 Terms and Types

In Table 1, we consider a typed version of polyadic Lπ extended with: (i) labeled values ℓ_v, called *variants* [19], with case analysis; (ii) tuple values $\langle\, v_1 .. v_n \,\rangle$, with pattern matching, (iii) constants k, called *keys*, with matching.

To deal with these rather complex values, we introduce several syntactic categories. As additional metavariables, we let s, p, q, r, m, t range over channels, y over variables, w range over values, Q over processes, and i, j, d, h, m over tuple, variant, or other indices. We abbreviate $\ell_\langle\rangle$ and $\ell_()$ with ℓ, as well as $\overline{q}\langle\rangle$ and $q().P$ with $\overline{q}$ and $q.P$, respectively, while $\widetilde{v}$ denotes a sequence $v_1 .. v_m$.

Restriction, both inputs, and both destructors are *binders* for the names $x, x_1, \ldots, x_m$ in the respective scopes $P, P_1, \ldots, P_m$. We assume the usual definitions of free and bound occurrences of names, based on these binders; the inductively defined functions $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ denote those of process P. Similarly, $\mathrm{fc}(P)$ and $\mathrm{bc}(P)$ denote the free and bound channels of process P. Moreover, $\mathrm{n}(P) = \mathrm{fn}(P) \cup \mathrm{bn}(P)$ and $\mathrm{c}(P) = \mathrm{fc}(P) \cup \mathrm{bc}(P)$. *Substitutions*, ranged over by σ, are type-preserving functions from names to values (types are introduced below). For an expression e, $e\sigma$ is the result of applying σ to e, with the usual renaming to avoid captures. We write $e\{{}^{v}/_{x}\}$ for a substitution that operates only on name x, leaving all the other names unchanged. *Relabelings*, ranged over by ρ, are functions from labels to labels. We write $P[{}^{\ell'}/_{\ell}]$ to replace all occurrences of label ℓ by label ℓ' in values occurring in term P. Substitution

Table 1. π-Calculus

Channels: $c \in \mathbf{C}$
Keys: $k \in \mathbf{K}$
Names: $\in \mathbf{N}$
$n ::= c \mid k$

Auxiliary: $u \in \mathbf{U}$
Variables: $\in \mathbf{X}$
$x ::= n \mid u$

Labels $\in \mathbf{L}$
$\ell, \ell_1, \ell_2, \ldots$

Values

$v ::= x$		variable
	$\mid \ell_v$	variant
	$\mid \langle\, v_1 .. v_n \,\rangle$	tuple

Types

$T ::= \mathbf{C}(T)$		channel type
	$\mid \mathbf{K}$	key type
	$\mid [\, \ell_1{:}T_1 \,; \ldots ; \ell_m{:}T_m \,]$	variant type
	$\mid \langle\, T_1 .. T_m \,\rangle$	tuple type
	$\mid X$	type variable
	$\mid \mu X.\, T$	recursive type

Processes

$P ::= \mathbf{0}$	nil process
$\mid\ c(x).P$	single *input*
$\mid\ \overline{c}v$	output
$\mid\ P_1 \mid P_2$	parallel
$\mid\ (\boldsymbol{\nu} n{:}T)\, P$	restriction
$\mid\ \mathbf{!}\, c(x).P$	replicated *input*
$\mid\ \mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3$	key testing
$\mid\ \mathsf{case}\ v\ \mathsf{of}\ \ell_1_(x_1){:}P_1 \,; \ldots ; \ell_m_(x_m){:}P_m$	variant *destructor*
$\mid\ \mathsf{let}\ (\, x_1 .. x_m \,) = v\ \mathsf{in}\ P$	tuple *destructor*

The *locality constraint* requires that in (*single* and *replicated*) *inputs* and (*variant* and *tuple*) *destructors* the bound names $x, x_1, \ldots, x_m$ must not be used in free input position within the respective scope $P, P_1, \ldots, P_m$.

and relabeling have the highest operator precedence, parallel composition the lowest.

Types are introduced for essentially three reasons: (i) they allow us to cleanly define some abbreviations, (ii) we use them to give a typed semantics of Øjeblik, and (iii) they allow us to formally prove the main result of the paper using typed behavioral equivalences. Abusing the notation for sets of names and the corresponding types, we use $\mathbf{K}$ and $\mathbf{C}$ also as type constructors, where channel types are parameterized over the type of value they carry. For variants and tuples we use standard notations (c.f. [19]). In a recursive type $\mu X.\, T$, occurrences of variable X in type T must be guarded, i.e., underneath a variant or channel constructor. We often omit the type annotation of restriction, when it is clear from the context or not important for the discussion. A *type environment* Γ is a finite mapping from variables to types. A *typing judgment* $\Gamma \vdash P$ asserts that process P is well-typed in Γ, and $\Gamma \vdash v{:}T$ that value v has type T in Γ. There is one typing rule for each process construct; each of them is straightforward. (We

provide the type system in the full version of the paper.) A type environment Γ is *closed* if it contains only names, no auxiliary variables.

2.2 Semantics and Proof Techniques

We equip our π-calculus with a standard *reduction semantics*. Its rules, defining the *reduction relation* $\rightarrow$, are precisely those of [13], but based on a notion of structural equivalence that is extended to deal with if-, case-, and let-constructs. For simplicity, we only consider the semantics of well-typed processes.

Definition 1. Structural equivalence, *written* $\equiv$, *is the smallest relation preserved by parallel composition and restriction, which satisfies the axioms below:*

- $P \equiv Q$, *if* P *is* α*-convertible to* Q;
- $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$;
- $(\boldsymbol{\nu} a)\,\mathbf{0} \equiv \mathbf{0}$, $(\boldsymbol{\nu} a)\,(\boldsymbol{\nu} b)\,P \equiv (\boldsymbol{\nu} b)\,(\boldsymbol{\nu} a)\,P$,
- $(\boldsymbol{\nu} a)\,(P \mid Q) \equiv P \mid (\boldsymbol{\nu} a)\,Q$, *if* $a \notin \mathrm{fn}(P)$;
- $\mathsf{if}\ [k_1{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \equiv P_1$;
- $\mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k_2{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \equiv P_2$, *if* $k_1{\neq}k$;
- $\mathsf{if}\ [k{=}k_1]\ \mathsf{then}\ P_1\ \mathsf{elif}\ [k{=}k_2]\ \mathsf{then}\ P_2\ \mathsf{else}\ P_3 \equiv P_3$, *if* $k_1{\neq}k{\neq}k_2$;
- $\mathsf{case}\ \ell_j_v_j\ \mathsf{of}\ \ell_1_(x_1){:}P_1\,;\ldots;\ell_j_(x_j){:}P_j\,;\ldots;\ell_m_(x_m){:}P_m \equiv P_j\{v_j/x_j\}$;
- $\mathsf{let}\ (\,x_1\,..\,x_m\,) = \langle\,v_1..\,v_m\,\rangle\ \mathsf{in}\ P \equiv P\{\widetilde{v}/\widetilde{x}\}$.

The relation $\Rightarrow$ is the reflexive-transitive closure of $\rightarrow$. For any relation $\mathcal{R}$ on processes, $\rightarrow_{\mathcal{R}}$ denotes $\rightarrow\mathcal{R}$, and $\Rightarrow_{\mathcal{R}}$ the reflexive-transitive closure of $\rightarrow_{\mathcal{R}}$.

As regards behavioral equivalences, we focus on *barbed bisimulation* [14], a uniform mechanism for defining behavioral equivalences in any process calculus possessing (i) a *reduction relation* and (ii) an *observability predicate*. Barbed bisimulation equips a global observer with a minimal ability to observe actions and/or process states but it is not a congruence. By closing barbed bisimulation under contexts we obtain a much finer relation. A context $C[\cdot]$ is a process with exactly one hole, written $[\cdot]$, where a process may be plugged in. A context $C[\cdot]$ is *static* if it is structurally equivalent to $(\boldsymbol{\nu}\widetilde{a})\,(P \mid [\cdot])$, for some P and $\widetilde{a}$. We let $E[\cdot]$ range over static contexts. In an *asynchronous* scenario, only output actions are usually considered observable [2]. A process P has a *barb* at channel c, written $P{\downarrow_c}$, if $P \equiv E[\,\bar{c}v\,]$ for some static context $E[\cdot]$, value v, and channel $c \in \mathrm{fn}(P)$. We write $P{\Downarrow_c}$, if there exists a process P' with $P \Rightarrow P'$ and $P'{\downarrow_c}$. In a *typed* scenario, only well-typed context are usually considered. Therefore, we recall the notion of (Δ/Γ)*-context* [17,19]: when filled with a process P with $\Gamma \vdash P$, a (Δ/Γ)-context $C[\cdot]$ guarantees the typing $\Delta \vdash C[P]$.

We often work with channels that have been extruded to the environment. To keep track of the fact that they cannot be used in input by the environment we generalize the standard typed barbed relations.

Definition 2 (Barbed Relations). *Let* $\mathcal{C} \subseteq \mathbf{C}$. Barbed $\mathcal{C}$-bisimilarity, *written* $\dot{\cong}_{\mathcal{C}}$, *is the largest symmetric relation on processes, such that* $P\ \dot{\cong}_{\mathcal{C}}\ Q$ *implies:*

1. *If* $P \xrightarrow{\tau} P'$, *then there exists* Q' *such that* $Q \Rightarrow Q'$ *and* $P'\ \dot{\cong}_{\mathcal{C}}\ Q'$

2. If $P{\downarrow_c}$, with $c \notin \mathcal{C}$, then $Q{\Downarrow_c}$.

Let Γ be a typing, and P and Q two processes such that $\Gamma \vdash P, Q$. We say that P and Q are barbed $\Gamma;\mathcal{C}$-equivalent*, written $P \simeq_{\Gamma;\mathcal{C}} Q$, if, for each closed type environment Δ and static (Δ/Γ)-context $C[\cdot]$ not containing names in $\mathcal{C}$ in input position, we have $C[P] \mathrel{\dot{\cong}}_{\mathcal{C}} C[Q]$. We say that P and Q are* barbed $\Gamma;\mathcal{C}$-congruent*, written $P \cong_{\Gamma;\mathcal{C}} Q$, if, for each closed type environment Δ and (Δ/Γ)-context $C[\cdot]$ not containing names in $\mathcal{C}$ in input position, we have $C[P] \mathrel{\dot{\cong}}_{\mathcal{C}} C[Q]$.*

If $\mathcal{C}{=}\emptyset$ in Definition 2, we omit $\mathcal{C}$ and get the standard definitions of barbed bisimilarity $\mathrel{\dot{\cong}}$, barbed Γ-equivalence $\simeq_\Gamma$, and barbed Γ-congruence $\cong_\Gamma$, respectively. If $\mathcal{C}{=}\{s\}$, we write $\cong_{\Gamma;s}$ for $\cong_{\Gamma;\mathcal{C}}$ and $\simeq_{\Gamma;s}$ for $\simeq_{\Gamma;\mathcal{C}}$. Due to the restrictions on the contexts, it holds that $\overline{s}v \cong_{\Gamma;s} \mathbf{0}$ and, by asynchrony, $s(u).\mathbf{0} \cong_{\Gamma;s} \mathbf{0}$.

The main inconvenience of barbed equivalences and congruences is that they use quantifications over contexts in the definition, and this unnecessarily complicates proofs of process equality. In the long version of this paper, we provide and make use of labeled characterizations. Parts of our proofs are based on the generalization (to our setting) of Lπ proof techniques [12] that are based on special processes called *link*. A link (sometimes called a *forwarder* [8]), is a process $!\,p(u).\overline{q}u$, abbreviated $p \triangleright q$, that behaves as an unbounded unordered buffer receiving values at one end (p) and retransmitting them at the other end (q). The following lemma allows us to always output bound names instead of free names. Its proof, in pure Lπ, can be found in [11], but its generalization to our typed setting is straightforward.

Lemma 1. *Let $\Gamma \vdash \overline{a}v$, for some Γ. Let $b \in \mathrm{fc}(v)$ with $\Gamma \vdash b{:}\mathbf{C}(T)$. Let $d \notin \mathrm{c}(v)$ and $w = v\{^d/_b\}$. Then $\overline{a}v \cong_\Gamma (\boldsymbol{\nu} d{:}\mathbf{C}(T))\,(\overline{a}w \mid d \triangleright b)$.*

3 Øjeblik: A Concurrent Object Calculus

The set $\mathcal{L}$ of untyped Øjeblik expressions is generated, as shown in Table 2, where l ranges over method *labels*. In this section, we present the Øjeblik's call-by-value semantics, first informally, then formalized using the π-calculus of § 2, through which also a standard behavioral semantics is defined.

Objects. An object $[\mathrm{l}_j{=}m_j]_{j\in J}$ consists of a finite collection of updatable named methods $\mathrm{l}_j{=}m_j$, for pairwise distinct labels l_j. In a method $\varsigma(s,\tilde{x})b$, the letter ς denotes a binder for the self variable s and argument variables $\tilde{x}$ within the body b. Moreover, every object in Øjeblik comes equipped with special methods for cloning, aliasing, surrogation, and pinging, which cannot be updated.

Method invocation $a.\mathrm{l}\langle\, a_1 \,..\, a_n \,\rangle$ with field l of the object a containing the method $\varsigma(s,\tilde{x})b$ results in the body b with the self variable s replaced by the enclosing object a, and the formal parameters $\tilde{x}$ replaced by the actual parameters $a_1 \,..\, a_n$ of the invocation. Method update $a.\mathrm{l}{\Leftarrow}m$ overwrites the current content of the named field l in a with method m and evaluates to the modified object. The operation $a.\mathsf{clone}$ creates an object with the same fields as the original object and initializes the fields to the same entries as in the original object.

a, b	$::= \mathbb{O}$	object
	$\mid a.\mathsf{l}\langle\, a_1 \,..\, a_n \,\rangle$	method invocation
	$\mid a.\mathsf{l} \Leftarrow m$	method update
	$\mid a.\mathsf{clone}$	shallow copy
	$\mid a.\mathsf{alias}\langle b \rangle$	object aliasing
	$\mid a.\mathsf{surrogate}$	object surrogation
	$\mid a.\mathsf{ping}$	object ping
	$\mid s, x, y, z$	variables
	$\mid \mathsf{let}\, x = a\, \mathsf{in}\, b$	local definition
	$\mid \mathsf{fork}\langle a \rangle$	thread creation
	$\mid \mathsf{join}\langle a \rangle$	thread destruction
$\mathbb{O}$	$::= [\mathsf{l}_j = m_j]_{j \in J}$	object record
m_j	$::= \varsigma(s_j, \tilde{x}_j) b_j$	method

Table 2. Øjeblik Syntax

The operation $a.\mathsf{alias}\langle b \rangle$ replaces the object a with an alias for b, regardless of whether a is already an alias or still an object record; if b itself is an alias to, e.g., an object c, then we consequently create, by transitivity, an *alias chain*.

The operation $a.\mathsf{surrogate}$ turns object a into a local proxy for a remote copy of itself, as implemented by the uniform method $\mathsf{surrogate} = \varsigma(s)s.\mathsf{alias}\langle s.\mathsf{clone} \rangle$. The operation $a.\mathsf{ping}$ is implemented by another uniform method $\mathsf{ping} = \varsigma(s)s$, such that it returns of the object o that results from the evaluation of a its "current identity", i.e., due to possible forwarding the current endpoint of an alias chain potentially starting at object o.

Scoping. An expression $\mathsf{let}\, x = a\, \mathsf{in}\, b$ (non-recursive) first evaluates a, binding the result to x, and then evaluates b within the scope of x.

Concurrency. Computational activity takes place within so-called *threads*. In addition to the main thread, new threads can be created by the fork command. The result of a forked computation is grabbed by the join command.

Self-Infliction, Serialization, Protection. The *current self* of a thread is the self of the last method invoked in the thread that has not yet completed. An Øjeblik operation is *self-inflicted* (also: *internal*) if it operates on the current self; otherwise, it is *external*. Øjeblik objects are *serialized*: within any object, at any time, at most one thread may be active. Serialization is implemented by associating with each object a *mutex* that is locked when a thread enters the object and released when the thread exits the object. More precisely, the mutex is always acquired for external operations, but never for internal ones; this concept allows a method to recursively call its siblings through self, but it excludes mutual recursion between objects. Øjeblik objects are *protected*: the *critical* operations update, aliasing, and cloning, are only allowed if they are internal. An operation is called *valid* if it is internal or not critical.

$$
\begin{array}{|rcl|}
\hline
[\![\, a.\mathsf{clone} \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,\Big(\; [\![\, a \,]\!]^k_q \,\big|\, q(y,k').\,\overline{y}\langle \mathsf{cln}_p, k'\rangle \;\Big) \\
[\![\, a.\mathsf{alias}\langle b\rangle \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q_x q_y)\,\Big(\; [\![\, a \,]\!]^k_{q_y} \,\big|\, q_y(y,k_y).([\![\, b \,]\!]^{k_y}_{q_x} \;\mid\; q_x(x,k_x)\,.\,\overline{y}\langle \mathsf{ali}_\langle x,p\rangle, k_x\rangle)\;\Big) \\
[\![\, a.\mathrm{l}_j \Leftarrow \varsigma(s,\tilde{x})b \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,\Big([\![\, a \,]\!]^k_q \,\big|\, q(y,k').(\boldsymbol{\nu} t)(\;!\,t(s,\widetilde{x},r,k).[\![\, b \,]\!]^k_r \,|\, \overline{y}\langle \mathsf{upd}_j_\langle t,p\rangle, k'\rangle)\Big) \\
[\![\, a.\mathrm{l}_j\langle\, a_1 \,..\, a_n \,\rangle \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q q_1 \cdots q_n)\,\Big(\; [\![\, a \,]\!]^k_q \,\big|\, q(y,k_0).([\![\, a_1 \,]\!]^{k_0}_{q_1} \mid q_1(x_1,k_1).([\![\, a_2 \,]\!]^{k_1}_{q_2} \mid \cdots \\
 & & \qquad\qquad q_n(x_n,k_n).\overline{y}\langle \mathsf{inv}_j_\langle x_1 \,..\, x_n, p\rangle, k_n\rangle \cdots))\Big) \\
[\![\, a.\mathsf{surrogate} \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,(\; [\![\, a \,]\!]^k_q \;\mid\; q(y,k')\,.\,\overline{y}\langle \mathsf{sur}_p, k'\rangle\;) \\
[\![\, a.\mathsf{ping} \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,(\; [\![\, a \,]\!]^k_q \;\mid\; q(y,k')\,.\,\overline{y}\langle \mathsf{png}_p, k'\rangle\;) \\
\hline
[\![\, \mathsf{let}\, x = a \,\mathsf{in}\, b \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,(\; [\![\, a \,]\!]^k_q \;\mid\; q(x,k')\,.\,[\![\, b \,]\!]^{k'}_p\;) \\
[\![\, x \,]\!]^k_p & \stackrel{\text{def}}{=} & \overline{p}\langle x,k\rangle \\
\hline
[\![\, \mathsf{fork}(a) \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q t)\,(\; [\![\, a \,]\!]^{\nu}_q \;\mid\; \overline{p}\langle t,k\rangle \;\mid\; q(x,k').t(r,k'').\overline{r}\langle x,k''\rangle\;) \\
[\![\, \mathsf{join}(b) \,]\!]^k_p & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} q)\,(\; [\![\, b \,]\!]^k_q \;\mid\; q(t,k')\,.\,\overline{t}\langle p,k'\rangle\;) \\
\hline
\end{array}
$$

Table 3. Øjeblik Semantics — Clients, Scoping, Concurrency

3.1 Translational Semantics

In addition to the core π-calculus of Section 2, we use *parameterized recursive definitions*, which can be faithfully represented in terms of replication [13].

The semantics as presented in Tables 3 and 4, maps Øjeblik-terms into π-calculus terms parameterized on two names: in a term $[\![\, a \,]\!]^k_p$, the channel p is used to return the term's result, while the key k represents the term's current self, which is required to deal with self-infliction. The essence of the semantics is to set up processes representing objects that serve clients' requests. Different requests for operating on objects are distinguished by corresponding π-calculus labels. We explain the semantics by showing how requests are generated by clients, and then how they are served by objects. We omit explanations for the semantics of scoping and concurrency; they can be found in the full paper.

Clients. In Table 3, the current self k of encoded terms is 'used' as the current self of the evaluation of the first subterm in left-to-right evaluation order. All the translations in Table 3 follow a common scheme. For example, in the translation of a method invocation $[\![\, a.\mathrm{l}_j\langle\, a_1 \,..\, a_n \,\rangle \,]\!]^k_p$, the subterms $a, a_1 \,..\, a_n$ have to be evaluated one after the other: the individual evaluations use private return channels $q, q_1 \,..\, q_n$, which are subsequently asked for the respective results $y, x_1 \,..\, x_n$, but also for the respective new current self $i, i_1 \,..\, i_n$ to be used by the next evaluation—this can be same as for the previous evaluation, but is not necessarily so (c.f. the description of object managers below). After the last subterm a_n has returned its result, the accumulated information is used to send a suitable request with label inv_j to self-channel y of object a, also carrying the overall result channel p and the latest current self i_n. Thus, the responsibility to signal a result on p is passed on to the respective object manager waiting at y.

$$[\![\,\mathbb{O}\,]\!]^k_p \stackrel{\text{def}}{=} (\boldsymbol{\nu} s\widetilde{t})\Big(\,\overline{p}\langle s,k\rangle \mid \mathsf{newO}_{\mathbb{O}}\langle\, s,\widetilde{t}\,\rangle \mid \prod_{j\in J} !\, t_j(s_j,\widetilde{x}_j,r,k').[\![\, b_j\,]\!]^{k'}_r\,\Big)$$

$$\mathsf{newO}_{\mathbb{O}}\langle\, s,\widetilde{t}\,\rangle \stackrel{\text{def}}{=} (\boldsymbol{\nu} m_{\mathrm{e}} m_{\mathrm{i}} k_{\mathrm{e}} k_{\mathrm{i}})\Big(\,\overline{m_{\mathrm{e}}} \mid \mathsf{OM}_{\mathbb{O}}\langle\, s,m_{\mathrm{e}},m_{\mathrm{i}},k_{\mathrm{e}},k_{\mathrm{i}},\widetilde{t}\,\rangle\,\Big)$$

$$\mathsf{newA}_{\mathbb{O}}\langle\, s,s_{\mathrm{a}}\,\rangle \stackrel{\text{def}}{=} (\boldsymbol{\nu} m_{\mathrm{e}} m_{\mathrm{i}} k_{\mathrm{e}} k_{\mathrm{i}})\Big(\,\overline{m_{\mathrm{e}}} \mid \mathsf{AM}_{\mathbb{O}}\langle\, s,m_{\mathrm{e}},m_{\mathrm{i}},k_{\mathrm{e}},k_{\mathrm{i}},s_{\mathrm{a}}\,\rangle\,\Big)$$

$$\begin{array}{l}
\mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k_{\mathrm{i}},\widetilde{t}\,\rangle \stackrel{\text{def}}{=} s(l,k).(\boldsymbol{\nu} k^*)\Big(\\
\quad \text{if } [k{=}k_{\mathrm{i}}] \text{ then}\\
\qquad \mathsf{case}\ l\ \mathsf{of}\ \mathsf{cln}_(r) : \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,\widetilde{t}\,\rangle \mid (\boldsymbol{\nu} s^*)\big(\,\overline{r}\langle s^*,k^*\rangle \mid \mathsf{newO}_{\mathbb{O}}\langle\, s^*,\widetilde{t}\,\rangle\,\big)\ ;\\
\qquad\quad \mathsf{ali}_(s_{\mathrm{a}},r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid \overline{r}\langle s_{\mathrm{a}},k^*\rangle\ ;\\
\qquad\quad \mathsf{upd}_j_(t',r) : \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,t_1\,..\,t_{j-1},t',t_{j+1}\,..\,t_n\,\rangle \mid \overline{r}\langle s,k^*\rangle\ ;\\
\qquad\quad \mathsf{inv}_j_(\widetilde{x},r) : \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,\widetilde{t}\,\rangle \mid \overline{t_j}\langle s,\widetilde{x},r,k^*\rangle\ ;\\
\qquad\quad \mathsf{sur}_(r) : \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,\widetilde{t}\,\rangle \mid [\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle\,]\!]^{k^*}_r\ ;\\
\qquad\quad \mathsf{png}_(r) : \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,\widetilde{t}\,\rangle \mid [\![\, s\,]\!]^{k^*}_r\\
\quad \text{elif } [k{=}k_{\mathrm{e}}] \text{ then}\\
\qquad \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,\widetilde{t}\,\rangle \,\big|\, \mathsf{case}\ l\ \mathsf{of}\ \mathsf{cln}_(r) : m_{\mathrm{i}}(k).\overline{m_{\mathrm{e}}}\ ;\\
\qquad\qquad\qquad \mathsf{ali}_(s_{\mathrm{a}},r) : m_{\mathrm{i}}(k).\overline{m_{\mathrm{e}}}\ ;\\
\qquad\qquad\qquad \mathsf{upd}_j_(t',r) : m_{\mathrm{i}}(k).\overline{m_{\mathrm{e}}}\ ;\\
\qquad\qquad\qquad \mathsf{inv}_j_(\widetilde{x},r) : \mathsf{CM}[\,\overline{t_j}\langle s,\widetilde{x},r^*,k^*\rangle\,]\ ;\\
\qquad\qquad\qquad \mathsf{sur}_(r) : \mathsf{CM}[\,[\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle\,]\!]^{k^*}_{r^*}\,]\ ;\\
\qquad\qquad\qquad \mathsf{png}_(r) : \mathsf{CM}[\,[\![\, s\,]\!]^{k^*}_{r^*}\,]\\
\quad \text{else } \mathsf{OM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k_{\mathrm{i}},\widetilde{t}\,\rangle \mid m_{\mathrm{e}}.\big(\,\overline{s}\langle l,k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k\,\big)\Big)
\end{array}$$

$$\mathsf{CM}[\cdot] \stackrel{\text{def}}{=} (\boldsymbol{\nu} r^*)\big(\,[\cdot] \mid r^*(y,k').m_{\mathrm{i}}(k'').\big(\,\overline{r}\langle y,k''\rangle \mid \overline{m_{\mathrm{e}}}\,\big)\big)$$

$$\begin{array}{l}
\mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k_{\mathrm{i}},s_{\mathrm{a}}\,\rangle \stackrel{\text{def}}{=} s(l,k).(\boldsymbol{\nu} k^*)\Big(\\
\quad \text{if } [k{=}k_{\mathrm{i}}] \text{ then}\\
\qquad \mathsf{case}\ l\ \mathsf{of}\ \mathsf{cln}_(r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid (\boldsymbol{\nu} s^*)\big(\,\overline{r}\langle s^*,k^*\rangle \mid \mathsf{newA}_{\mathbb{O}}\langle\, s^*,s_{\mathrm{a}}\,\rangle\,\big)\ ;\\
\qquad\quad \mathsf{ali}_(s'_{\mathrm{a}},r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s'_{\mathrm{a}}\,\rangle \mid \overline{r}\langle s'_{\mathrm{a}},k^*\rangle\ ;\\
\qquad\quad \mathsf{upd}_j_(t',r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid \overline{s_{\mathrm{a}}}\langle l,k\rangle\ ;\\
\qquad\quad \mathsf{inv}_j_(\widetilde{x},r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid \overline{s_{\mathrm{a}}}\langle l,k\rangle\ ;\\
\qquad\quad \mathsf{sur}_(r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid \overline{s_{\mathrm{a}}}\langle l,k\rangle\ ;\\
\qquad\quad \mathsf{png}_(r) : \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid \overline{s_{\mathrm{a}}}\langle l,k\rangle\\
\quad \text{elif } [k{=}k_{\mathrm{e}}] \text{ then } \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k^*,s_{\mathrm{a}}\,\rangle \mid m_{\mathrm{i}}(k).\big(\,\overline{s_{\mathrm{a}}}\langle l,k\rangle \mid \overline{m_{\mathrm{e}}}\,\big)\\
\quad \text{else} \qquad\qquad \mathsf{AM}_{\mathbb{O}}\langle\, s,\widetilde{m},k_{\mathrm{e}},k_{\mathrm{i}},s_{\mathrm{a}}\,\rangle \mid m_{\mathrm{e}}.\big(\,\overline{s}\langle l,k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k\,\big)\Big)
\end{array}$$

Table 4. Øjeblik Semantics — Objects

Objects. The semantics $[\![\, \mathbb{O} \,]\!]_p^k$ of an object $\mathbb{O} := [\mathrm{l}_j{=}\varsigma(s_j, \tilde{x}_j)b_j]_{j\in J}$, as shown in Table 4 (again similar to [10]), consists of a message that returns the object's reference s together with the current self k on channel p, a composition of replicated processes that give access to the method bodies $[\![\, b_j \,]\!]_r^{k'}$, and a new object process $\mathsf{newO}_{\mathbb{O}}\langle\, s, \widetilde{t} \,\rangle$ that connects invocations on s from the outside to the method bodies, which are invoked by the trigger names $\widetilde{t}$. Inside $\mathsf{newO}_{\mathbb{O}}\langle\, s, \widetilde{t} \,\rangle$, several private names are needed: *mutexes* $\widetilde{m} := m_{\mathrm{e}}, m_{\mathrm{i}}$ are used for serialization; the (*internal*) key k_{i} is used to detect self-infliction; the (*external*) key k_{e} is used to implement serialization in a concurrent environment (see later on). The behavior of objects and aliases is represented by the object manager OM and alias manager AM, respectively, which both, for each request arriving along their reference s, first check for self-infliction $[k{=}k_{\mathrm{i}}]$, and then, by simple case analysis, for the kind of the request. We first explain how internal requests are served in objects and aliases. External requests will be served later.

Serving Internal Requests $[k{=}k_{\mathrm{i}}]$ No serialization or protection is required.

Object Managers (OM). For each field, the manager may activate appropriate instances of method bodies (case inv_j: the method body bound to l_j along trigger name t_j) and administer updates (case upd_j: install a new trigger name t'). Cloning (case cln) restarts the current object manager in parallel with a new object, which uses the same method bodies $\widetilde{t}$, but is accessible through a fresh reference s^*. Aliasing (case ali) starts an appropriate alias manager AM instead of re-starting the previous object manager OM. Surrogation and ping (cases sur and png) are modeled according to their uniform method definitions.

Alias Managers (AM). Local requests for cloning and aliasing are allowed and behave analogous to the respective clauses in object managers, but restarting AM instead of OM. Update, invocation, surrogation, and ping requests are immediately forwarded *as is* to the aliasing target s_{a}.

Nonces (k^*). To guarantee the receptiveness of objects, managers OM and AM always have to be restarted with some possibly changed state. However, serialization requires that at any moment, only one request shall be *active* in an object. According to our semantics, program contexts will never give rise to several competing self-inflicted requests, but, when reasoning within arbitrary π-calculus contexts, as we do in § 4.2, their existence must be taken into account. Therefore, we add another layer of protection to increase the robustness of serialization: each time a request enters a manager, a fresh key k^* is created to be used in the restarted manager; this key must subsequently be used as the current self for all activities enabled by the current request. Thus, the consumption of one of several competing pending requests renders its competitors external. Note that nonces would not be necessary if we were interested in correct behaviors within only translated contexts.

Serving External Requests $[k{\neq}k_{\mathrm{i}}]$ Serialization and protection are required.

In order to clarify the behavior of an object manager serving an external request, Figure 1 show the five relevant "states", starting from a *free* manager

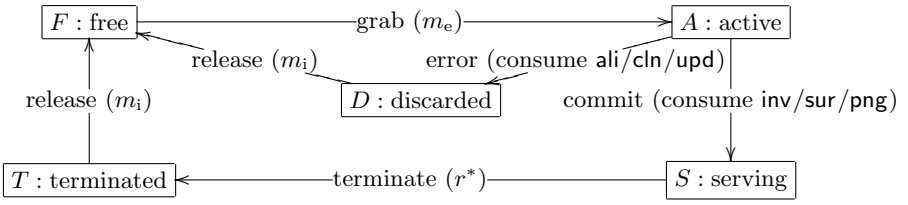


Fig. 1. Object Manager Serving External Requests

that becomes *active* by some pending request grabbing its serialization-lock. Then, if the request is protection-critical it is *discarded*, otherwise the manager *commits* to it and *serves* it until explicit *termination*. In both cases, the manager becomes free again by releasing the lock. Note that internal requests can only be served in "state" S of a manager that currently already serves some request. In Subsection 3.3, we explain Figure 1 in more detail; for now, it just offers an informal device to guide the explanations of the semantics of object managers.

Serialization ($m_\mathrm{e}, m_\mathrm{i}, k_\mathrm{e}$). As mentioned earlier, mutual exclusion within an object is implemented by mutexes, so, upon creation of a new object newO, the fresh mutex channel m_e is initialized. According to serialization, the intended continuation behavior of an incoming external requests is blocked on m_e, once it enters a manager. The manager itself is immediately restarted and remains receptive; it also remains in its "state" according to Figure 1. Arbitrarily many requests can be blocked this way and compete for the mutex m_e once it becomes available. A successfully unblocked request is resent to the same manager, but now carrying another key k_e, which allows the manager to detect that the request has grabbed the mutex, so the manager can evolve into "state" A. We call *pre-processing* the procedure of intermediate blocking of requests and their subsequent reemission with key k_e instead. Alongside with the pre-processed request, its former current self k is stored on the (internal) mutex m_i for recovery after termination. This recovery is actually necessary since the original current self k is possibly required for use later on by the sender of the request.

Nonces (k^*). Pre-processing must not reinitialize the key k_i of the restarted manager: a currently self-inflicted operation interleaved by pre-processing might be hindered to proceed, because it could unintendedly become external.

Object Managers (OM). Cloning, aliasing, or update, are critical operations. Once a respective pre-processed request is consumed, the manager evolves from "state" A into "state" D: the request and its former current self k, stored on channel m_i, will be simply ignored when releasing $\overline{m_\mathrm{e}}$ by consuming $\overline{m_\mathrm{i}}k$.

Invocation, surrogation, and ping, are non-critical operations. Once a respective pre-processed request is consumed, the manager evolves from "state" A into "state" S implying that no other external request shall be served (apart from pre-processing) until the current one has terminated. In order to be notified of that event, we employ a *call manager* protocol, represented by the context $\mathsf{CM}[\cdot]$: instead of delegating to some other process the responsibility of returning a result

$$
\begin{array}{rll}
A, B & ::= [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J} & \text{object record type} \\
 & \mid \mathrm{Thr}(A) & \text{thread type}
\end{array}
$$

$$
\begin{array}{rl}
\mathbf{R}(X) & \stackrel{\text{def}}{=} \mathbf{C}(X, \mathbf{K}) \\
\mathbf{M}(B_1 \,..\, B_n{\to}\widehat{B}) & \stackrel{\text{def}}{=} [\![\, B_j \,]\!] \,..\, [\![\, B_n \,]\!], \mathbf{R}([\![\, \widehat{B} \,]\!]) \\
[\![\, [\mathrm{l}_j{:}\widetilde{B}_j{\to}\widehat{B}_j]_{j\in J} \,]\!] & \stackrel{\text{def}}{=} \\
\multicolumn{2}{c}{\mu X.\, \mathbf{C}(\langle \left[\begin{array}{ll}
\mathsf{cln} & : \quad \mathbf{R}(X) \\
\mathsf{ali} & : \quad \langle\, X, \mathbf{R}(X) \,\rangle \\
\mathsf{upd}_j & : \langle\, \mathbf{C}(\langle\, X, \mathbf{M}(\widetilde{B}_j{\to}\widehat{B}_j), \mathbf{K} \,\rangle, \mathbf{R}(X)) \,\rangle \\
\mathsf{inv}_j & : \quad \langle\, \mathbf{M}(\widetilde{B}_j{\to}\widehat{B}_j) \,\rangle \\
\mathsf{sur} & : \quad \mathbf{R}(X) \\
\mathsf{png} & : \quad \mathbf{R}(X)
\end{array}\right]_{j\in J}, \mathbf{K} \,\rangle)} \\
[\![\, \mathrm{Thr}(A) \,]\!] & \stackrel{\text{def}}{=} \mathbf{C}(\langle\, \mathbf{R}(A), \mathbf{K} \,\rangle) \\
[\![\, \Gamma, x{:}A \,]\!] & \stackrel{\text{def}}{=} [\![\, \Gamma \,]\!], x : [\![\, A \,]\!]
\end{array}
$$

Table 5. Translation of Øjeblik-types

on r, a fresh return channel r^* is created to be used within $[\cdot]$ in place of r, such that the result will first appear on r^*. Until this event, other external requests remain blocked, while internal request may well be served. After this event, the manager evolves from "state" S into "state" T, where the former current self can be grabbed from m_i, the result y be forwarded to the intended result channel r (along with the former current self), and the mutex m_e be released. Note that externally triggered method bodies $[\![\, b_j \,]\!]$, and also surrogation and ping bodies $[\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle \,]\!]$ and $[\![\, s \,]\!]$, are all run in the context of the nonce k^*, which is now the internal key of the OM, so their further calls to s will be self-inflicted. This is essential for surrogation, since cloning and aliasing are only allowed internally.

Alias Managers (AM). According to our discussion in [16], external requests that arrive at an active alias manager should be blocked until the current activity finishes and the lock m_e is released. Once this happens, all external requests are—after three intermediate steps via channels m_e, s, and m_i—forwarded to the aliasing target s_a. The pre-processing of requests, presumably superfluous in alias managers, is necessary also there, because there may be pending pre-processed requests that have come to existence when s was connected to an OM.

Type Translation Øjeblik is equipped with a standard static type system [16]. The translation of terms into our π-calculus is accompanied with a straightforward translation of the corresponding types in Table 5. Its importance is that (i) the two translations together preserve typings (see the full paper for details), and (ii) that we can exploit the type information in proofs of properties of Øjeblik-terms, not least by applying typed barbed relations.

3.2 Behavioral Semantics

A standard way to define *program equivalence* is to compare the behavior of programs within arbitrary program contexts, as, for example, shown in previous work on the Imperative Object Calculus (IOC) [1,6]. In our setting, an *Øjeblik context* $C[\cdot]$ has a single hole $[\cdot]$ that may be filled with an Øjeblik term. In the remainder of the paper, we assume that Øjeblik-contexts always yield well-typed terms when plugging some Øjeblik-term into the hole. As a simple notion of program behavior to be tested based on our Øjeblik-semantics, we choose the existence of barbs as in § 2. This closely follows the intuition that a term $[\![\, a \,]\!]^k_p$ should tell its result on name p as soon as it knows it. So, an Øjeblik term *converges*, if its semantics *may* report its result on the name p.

Definition 3 (Convergence). *If $a \in \mathcal{L}$ is an Øjeblik term, then $a{\Downarrow}$ if $[\![\, a \,]\!]^k_p{\Downarrow_p}$.*

Definition 4 (Behavioral equivalence). *Two Øjeblik terms $a, b \in \mathcal{L}$ are* behaviorally equivalent*, written $a \doteq b$, if $C[a] \Downarrow$ iff $C[b] \Downarrow$ for all contexts $C[\cdot]$.*

Note that this equivalence is based on a *may*-variant of convergence. With respect to our goal of reasoning about surrogation, *must*-variants of convergence would be too strong, because, in a concurrent language with fork, threads may nondeterministically affect the outcome and convergence of evaluation.

3.3 Properties of the Translational Semantics

An important advantage of using a typed Lπ semantics is the easy provability that object managers are the unique receptors for their (bound) self-channels.

Lemma 2 (Uniqueness of Objects). *Let a be an Øjeblik term. If $[\![\, a \,]\!]^k_p \Rightarrow Z$ with $Z \equiv (\boldsymbol{\nu}\widetilde{z})\,(\, M \mid \mathsf{OM}_{\mathbb{O}}\langle\, s, \ldots \,\rangle\,)$ or $Z \equiv (\boldsymbol{\nu}\widetilde{z})\,(\, M \mid \mathsf{AM}_{\mathbb{O}}\langle\, s, \ldots \,\rangle\,)$, then $s \in \widetilde{z}$ and s does not appear free in input position within M.*

We now analyze, referring to Figure 1, how the shape of a particular object manager and its surrounding context evolves during computation. We will need a particular case thereof (Lemma 3) later on in the proof of Theorem 2.

Observation 1: Pre-processing does not change the "state" of object managers. At any time, an object/alias manager is ready to receive a request $\overline{s}\langle l, k\rangle$ with $k_{\mathrm{e}} \neq k \neq k_{\mathrm{i}}$. The manager is identically restarted afterwards, but will have spawned a process $m_{\mathrm{e}}.(\, \overline{s}\langle l, k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k \,)$ that replaces the consumed request. Let us assume requests $\overline{s}v_j$ with $v_j := \langle\, l_j, k_j \,\rangle$ for $1 \leq j \leq h$ (and $\widetilde{v} := v_1 \ldots v_h$) are pre-processed by the object manager $\mathsf{OM}_{\mathbb{O}}\langle\, s, m_{\mathrm{e}}, m_{\mathrm{i}}, k_{\mathrm{e}}, k_{\mathrm{i}}, \widetilde{t}\,\rangle$, so $k_{\mathrm{e}} \neq k_j \neq k_{\mathrm{i}}$ for all $1 \leq j \leq h$. Then:

$$\mathsf{PP}_{\mathbb{O}}\langle\, s, m_{\mathrm{e}}, m_{\mathrm{i}}, k_{\mathrm{e}}, \widetilde{v}\,\rangle \stackrel{\text{def}}{=} \prod_{1 \leq j \leq h} m_{\mathrm{e}}.(\, \overline{s}\langle l_j, k_{\mathrm{e}}\rangle \mid \overline{m_{\mathrm{i}}}k_j \,)$$

Observation 2: In object managers, k_i may be extruded, $m_\mathrm{e}, m_\mathrm{i}, k_\mathrm{e}$ may not. Assume that an inv_j-request along s appears at $\mathsf{OM}_\mathbb{O}\langle\, s, m_\mathrm{e}, m_\mathrm{i}, k_\mathrm{e}, k_\mathrm{i}, \widetilde{t}\,\rangle$, is pre-processed, gets the mutex m_e and re-enters along s with k_e. During this, according to the semantics, a fresh internal key k^* is created and extruded to the corresponding method body. The names $\widetilde{n} := m_\mathrm{e}, m_\mathrm{i}, k_\mathrm{e}$ are never extruded; they constitute the proper boundary of a manager during computation. This observation also provides the formal basis for Figure 1: a term Z containing an object manager $\mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k_\mathrm{i}, \widetilde{t}\,\rangle$ corresponds to "state" F, A, D, S, or T, respectively (for this manager), if after moving—as far as possible—all top-level parallel components of Z outside the scope of $(\boldsymbol{\nu}\widetilde{n})$ the remaining component of Z inside the scope has a characteristic shape. In the full paper, we show the complete analysis; here, we outline two special cases: *free* object managers in "state" F, and *committing* object managers ready to evolve from "state" A into "state" S.

Observation 3: An object manager is free*, if its external mutex is available.* In our semantics, a manager is willing to grant access, if the external mutex m_e occurs unguarded in the term that describes the current "state", so the general shape of a *free object* (and analogously *alias*) *manager* is:

$$\begin{aligned}\mathsf{freeO}_\mathbb{O}\langle\, s, k_\mathrm{i}, \widetilde{t}, \widetilde{v}\,\rangle &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{n})\big(\,\overline{m_\mathrm{e}} \;\big|\; \mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k_\mathrm{i}, \widetilde{t}\,\rangle \;\big|\; \mathsf{PP}_\mathbb{O}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle\,\big)\\ \mathsf{freeA}_\mathbb{O}\langle\, s, k_\mathrm{i}, s_\mathrm{a}, \widetilde{v}\,\rangle &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{n})\big(\,\overline{m_\mathrm{e}} \;\big|\; \mathsf{AM}_\mathbb{O}\langle\, s, \widetilde{n}, k_\mathrm{i}, s_\mathrm{a}\,\rangle \;\big|\; \mathsf{PP}_\mathbb{O}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle\,\big)\end{aligned}$$

where the keys mentioned in $\widetilde{v}$ of $\mathsf{PP}_\mathbb{O}\langle\ldots\rangle$ neither match k_e nor k_i. Note that $\mathsf{newO}_\mathbb{O}\langle\, s, \widetilde{t}\,\rangle \equiv (\boldsymbol{\nu}k_\mathrm{i})\,\mathsf{freeO}_\mathbb{O}\langle\, s, k_\mathrm{i}, \widetilde{t}, \emptyset\,\rangle$, and analogously for $\mathsf{newA}_\mathbb{O}\langle\ldots\rangle$.

Observation 4: An object manager is ready to commit*, if it may consume a valid pre-processed request.* The following lemma derives from the ability to commit to a valid external request—visible as the availability of a valid pre-processed request, i.e., a request carrying k_e—the shape of the object manager before and after commitment, including all of its current pre-processed requests.

Lemma 3 (Committing Object Manager). *Let $a \in \mathcal{L}$ and $[\![\, a \,]\!]^k_p \;\Rightarrow Z$. If $Z \equiv E[\,\overline{s}\langle l, k_\mathrm{e}\rangle \mid \mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k_\mathrm{i}, \widetilde{t}\,\rangle\,]$ with $l \in \{\mathsf{inv}_j_\langle\widetilde{x}, r\rangle, \mathsf{png}_r, \mathsf{sur}_r\}$, and $\widetilde{n} = m_\mathrm{e}, m_\mathrm{i}, k_\mathrm{e}$, then $Z \;\rightarrow Z'$ for*

$$\begin{aligned}Z &\equiv \widehat{E}[\,(\boldsymbol{\nu}\widetilde{n})\big(\,\overline{m_\mathrm{i}}k' \mid \mathsf{PP}_\mathbb{O}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \mid \mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k_\mathrm{i}, \widetilde{t}\,\rangle \mid \overline{s}\langle l, k_\mathrm{e}\rangle\,\big)\,]\\ Z' &\equiv \widehat{E}[\,(\boldsymbol{\nu}\widetilde{n})\big(\,\overline{m_\mathrm{i}}k' \mid \mathsf{PP}_\mathbb{O}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \mid (\boldsymbol{\nu}k^*)\big(\,\mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle \mid \mathsf{CM}[\,X_l\langle\, s\,\rangle^{k^*}_{r^*}\,]\,\big)\big)\,]\end{aligned}$$

for some key k', some set $\widetilde{v}$ of pre-processed requests, and $X_l\langle\, s\,\rangle$ denoting the respective continuation behavior of Table 4.

Note that the k^* in Z' is fresh, so it can be extruded over $\mathsf{PP}_\mathbb{O}\langle\ldots\rangle$ and $\overline{m_\mathrm{i}}k'$. As special cases, for $l \in \{\mathsf{png}_r, \mathsf{sur}_r\}$, of *committed* object managers, we define

$$\begin{aligned}F[\cdot] &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{n}k^*)\big(\,\overline{m_\mathrm{i}}k \big|\, \mathsf{PP}_\mathbb{O}\langle\, s, \widetilde{n}, \widetilde{v}\,\rangle \;\big|\; \mathsf{OM}_\mathbb{O}\langle\, s, \widetilde{n}, k^*, \widetilde{t}\,\rangle \;\big|\; [\cdot]\,\big)\\ \mathsf{pingO}_\mathbb{O}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle &\stackrel{\text{def}}{=} F[\mathsf{CM}[\,[\![\, s \,]\!]^{k^*}_{r^*}\,]]\\ \mathsf{surO}_\mathbb{O}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle &\stackrel{\text{def}}{=} F[\mathsf{CM}[\,[\![\, s.\mathsf{alias}\langle s.\mathsf{clone}\rangle \,]\!]^{k^*}_{r^*}\,]]\end{aligned}$$

and discuss their properties in Section 4.2.

4 On the Safety of Surrogation

In [16], we motivated the equation $a.\mathsf{ping} \doteq a.\mathsf{surrogate}$ for contextual equivalence $\doteq$ based on convergence as a valuable goal for proving safety of surrogation. Its interpretation is that an object a, if it is responsive to a ping-request, behaves the same as the object a after surrogation. One of the main observations in [16] was that the desired equation can not hold in full generality for Øjeblik-contexts $C[\cdot]$, in which the operation $x.\mathsf{surrogate}$ could occur internally. The reason is that, after *internal* surrogation, an object may misuse by intention the old and new references to *itself*. Actually, the advice to avoid internal surrogation is analogous to the fact that programmers, knowing that $x{=}0$, should never use division by x. In constrast, external surrogation corresponds to the case where a program receives x from some other module, so it should be guaranteed that x will never be 0. Analogously, we conjectured in [16] that in our semantics *external* surrogation is guaranteed to be safe.

In this section, we prove that $C[x.\mathsf{ping}]{\Downarrow}$ iff $C[x.\mathsf{surrogate}]{\Downarrow}$ for precisely those cases where $C[\cdot]$ will never lead to self-inflicted occurrences of $x.\mathsf{surrogate}$. Although this is an undecidable criterion [4], we may still formalize it in terms of our π-calculus semantics, as we do in Subsection 4.1, for its use in formal proofs. In Subsection 4.2, we study the behavior of objects before and after surrogation within tightly restricted contexts and prove them to be barbed equivalent. In Subsection 4.3, we then give an outline of the formal proof for the safety of external surrogations. The full paper also offers a static type system that guarantees that surrogations will never be internal. The full paper also offers a static type system that guarantees that surrogations will never be internal.

4.1 On the Absence of Internal Surrogation

Here, we study how to formalize that $C[\cdot]$ will never lead to self-inflicted occurrences of the term $x.\mathsf{surrogate}$, when plugged into the hole.

Recall from the Øjeblik semantics in § 3 that in a particular state $[\![\, a \,]\!]^k_p \;\Rightarrow Z$ in the computation of an arbitrary Øjeblik term a, a particular sur-request is self-inflicted, if $Z \equiv E[\; \overline{s}\langle \mathsf{sur}_r, k\rangle \mid \mathsf{OM}_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, \widetilde{t}\,\rangle \;]$ with $k{=}k_{\mathrm{i}}$, because it is ready to enter the OM with $k{=}k_{\mathrm{i}}$ (c.f. Table 4). Since we must ensure that a sur-request *never* leads to internal surrogation, we must quantify over all derivatives of $[\![\, a \,]\!]^k_p$ and check for self-infliction in each of them.

Note that, starting from the term $[\![\, C[x.\mathsf{surrogate}] \,]\!]^k_p$, we should not be concerned with arbitrary sur-requests that appear at top-level during computation, but only with those that "arise from the request in the hole". However, this property is hard to determine for two different reasons: (1) *All* of the names mentioned in a sur-request may be changed dynamically by instantiation: s (due to forwarding), r (due to a call manager protocol), and k (due to pre-processing). (2) We have to consider arbitrarily many duplications of the request in the case that the hole appears, at the level of Øjeblik terms, within in a method body, which leads to replication in the π-calculus semantics. For both reasons, we need a tool to uniquely identify the various incarnations of the request.

Let $\mathsf{operate} \in \{\mathsf{ping}, \mathsf{surrogate}\}$, and let $\mathsf{op} \in \{\mathsf{png}, \mathsf{sur}\}$ denote the corresponding π-calculus label (c.f. Table 3). We introduce the *additional* Øjeblik labels $\mathsf{operate}^\star \in \{\mathsf{ping}^\star, \mathsf{surrogate}^\star\}$, writing $\mathcal{L}_T$ for the resulting language. The intuition is that tagged labels are semantically treated exactly like their untagged counterparts, but can syntactically be distinguished from them. Consequently, we give a tagged semantics, written $[\![\ ]\!]$, by adding the respective clauses for tagged labels, which are just copies of the clauses for the untagged labels; we use the tagged π-calculus labels $\mathsf{op}^\star \in \{\mathsf{png}^\star, \mathsf{sur}^\star\}$ at the semantics level. As a result, both tagged and untagged requests can be sent to object and alias managers; object managers ignore the tagging information of requests and treat $\mathsf{op}^\star$-and op-requests identically, but alias managers preserve the tagging information since they simply forward requests. We also add a tag to all parameterized definitions and abbreviations when considering the tagged semantics.

The semantics is not affected by including tagging information.

Lemma 4. *Let x be an Øjeblik variable and $C[\cdot]$ an untagged Øjeblik context. Then: $C[x.\mathsf{operate}]{\Downarrow}$ iff $[\![C[x.\mathsf{operate}^\star]]\!]^k_p{\Downarrow_p}$.*

However, tagging helps us to detect all "requests arising from the hole".

Definition 5 (Safe Contexts). *Let x be a variable and $C[\cdot]$ an untagged Øjeblik context. Then, $C[\cdot]$ is called* safe *for $x.\mathsf{surrogate}$, if $[\![C[x.\mathsf{surrogate}^\star]]\!]^k_p \Rightarrow_{\equiv} E[\ \overline{s}\langle \mathsf{sur}^\star_r, k\rangle \mid \mathsf{OM}^\star_{\mathbb{O}}\langle\, s, \widetilde{m}, k_{\mathrm{e}}, k_{\mathrm{i}}, \widetilde{t}\,\rangle\ ]$ implies that $k \neq k_{\mathrm{i}}$.*

We replay the definition using ping instead of $\mathsf{surrogate}$. By definition of the semantics, an Øjeblik context $C[\cdot]$ is then safe for $x.\mathsf{surrogate}$ if and only if it is safe for $x.\mathsf{ping}$. For convenience, by abuse, we simply call $C[\cdot]$ to be *safe for x*.

4.2 Committed External Surrogation is Transparent

Our main result focuses on external surrogations. In the following we show that the two versions of an object manager at s that are committed to an external png- and sur-request, respectively (c.f.§ 3.3), are related by typed barbed equivalence.

Theorem 1. *Let $\Gamma \vdash \mathsf{surO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle$ and $\Gamma \vdash \mathsf{pingO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle$. Then:*

$$\mathsf{surO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle \simeq_{\Gamma;s} \mathsf{pingO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle.$$

The proof of Theorem 1 requires several strong lemmas. Lemma 5 proves that surrogation results in an alias pointing to a clone of the old object. Its proof heavily relies on the nonces (c.f. page 399) used in the implementation of object and alias managers, which control the interference with the environment. Lemma 6 proves that the aliased object manager appearing in Lemma 5 behaves as a forwarder. Lemma 7 uses Lemma 1 to prove correctness of inlining. Lemma 8 proves that pre-processing external requests does not preclude other requests. Lemma 9 involves two confluent reductions from right to left along r^* and m_{i}, respectively. Finally, Theorem 1 can be established by applying the previous lemmas.

Lemma 5. *If Γ is a suitable type environment for the processes below, then:*

$$\mathsf{surO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle \simeq_{\Gamma;s} (\boldsymbol{\nu} s^*)\big((\boldsymbol{\nu} k_{\mathrm{i}})\,\mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \widetilde{v}\,\rangle \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s^*, \widetilde{t}\,\rangle \,\big|\, \overline{r}\langle s^*, k\rangle\big).$$

Lemma 6. *Let $\widetilde{v} := v_1..v_n$, and $v_j{:=}\langle\, l_j, k_j\,\rangle$ for $1{\leq}j{\leq}n$. If Γ is a suitable type environment for the processes below, then:*

$$(\boldsymbol{\nu} k_{\mathrm{i}})\,\mathsf{freeA}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, s^*, \widetilde{v}\,\rangle \simeq_{\Gamma;s} s \triangleright s^* \,\Big|\, \prod_{1\leq j\leq n} \overline{s^*} v_j.$$

Lemma 7. *Let P be a process and s a channel such that $s \notin \mathrm{fc}(P)$. If Γ is a suitable type environment for the processes below, then:*

$$(\boldsymbol{\nu} s^*)\,\big(\, s \triangleright s^* \mid P\,\big) \simeq_{\Gamma;s} P\{{}^{s}/_{s^*}\}.$$

Lemma 8. *Let $\widetilde{v} := v_1..v_n$ with $v_j{:=}\langle\, l_j, k_j\,\rangle$ and $k_j{\neq}k_{\mathrm{i}}$ for $1{\leq}j{\leq}n$. If Γ is a suitable type environment for the processes below, then:*

$$\prod_{1\leq j\leq n} \overline{s} v_j \,\big|\, \mathsf{newO}_{\mathbb{O}}\langle\, s, \widetilde{t}\,\rangle \simeq_{\Gamma;s} (\boldsymbol{\nu} k_{\mathrm{i}})\,\mathsf{freeO}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, \widetilde{t}, \widetilde{v}\,\rangle.$$

Lemma 9. *Let $\widetilde{v} := v_1..v_n$ with $v_j{:=}\langle\, l_j, k_j\,\rangle$ and $k_j{\neq}k_{\mathrm{i}}$ for $1{\leq}j{\leq}n$. If Γ is a suitable type environment for the processes below, Then:*

$$\overline{r}\langle s, k\rangle \,\big|\, (\boldsymbol{\nu} k_{\mathrm{i}})\,\mathsf{freeO}_{\mathbb{O}}\langle\, s, k_{\mathrm{i}}, \widetilde{t}, \widetilde{v}\,\rangle \simeq_{\Gamma} \mathsf{pingO}_{\mathbb{O}}\langle\, s, r, k, \widetilde{t}, \widetilde{v}\,\rangle.$$

4.3 External Surrogation is Safe

We prove our main theorem that $x.\mathsf{ping} \doteq x.\mathsf{surrogate}$ for safe contexts $C[\cdot]$.

Theorem 2 (Safety). *Let x be an object variable and $C[\cdot]$ an untagged context in Øjeblik. If $C[\cdot]$ is safe for x, then $C[x.\mathsf{ping}]{\Downarrow}$ iff $C[x.\mathsf{surrogate}]{\Downarrow}$.*

Proof. (Sketch) By Lemma 4, our proof obligation is equivalent to:

$$[\![\, C[x.\mathsf{ping}^\star] \,]\!]^k_p{\Downarrow_p} \;\text{ iff }\; [\![\, C[x.\mathsf{surrogate}^\star] \,]\!]^k_p{\Downarrow_p}.$$

This allows us to make use of the assumption on the safety of context $C[\cdot]$.

Since the semantics $[\![\ ]\!]$ is compositional, there is a π-calculus context $D[\cdot]$ and names y, j, q, such that $[\![\, C[x.\mathsf{operate}^\star] \,]\!]^k_p = D[\overline{y}\langle \mathsf{op}^\star_q, j\rangle]$, where $D[\cdot]$ itself does not contain any message carrying a tagged request. We prove that

$$D[\overline{y}\langle \mathsf{png}^\star_q, j\rangle]{\Downarrow_p} \;\text{ iff }\; D[\overline{y}\langle \mathsf{sur}^\star_q, j\rangle]{\Downarrow_p}.$$

and concentrate on the implication from right to left. The converse is analogous.

Assume that $D[\overline{y}\langle \mathsf{sur}^\star_q, j\rangle] \Downarrow_p$. If $D[N]{\Downarrow_p}$ for every process N, then this is also the case for $N = \overline{y}\langle \mathsf{png}^\star_q, j\rangle$; otherwise, the $\mathsf{sur}^\star$-request must contribute to the barb. Therefore, we assume $D[\overline{y}\langle \mathsf{sur}^\star_q, j\rangle] \;\Rightarrow P \downarrow_p$ and show that there is

a corresponding $D[\overline{y}\langle \mathsf{png}^\star_q, j\rangle] \Rightarrow_{\simeq_\Gamma} Q \downarrow_p$ where $Q = P[{}^{\mathsf{png}^\star}/_{\mathsf{sur}^\star}]$. Since typed barbed equivalence $\simeq_\Gamma$ and relabeling preserve convergence, this suffices.

We distinguish between insignificant and significant transitions, where the former can be mimicked easily, while the latter require some work. Recall that a reduction step due to an external request is *committing* (c.f. § 3.3), if it represents the consumption of a pre-processed request by an object manager. Now, we combine this characterization with the fact that we have to concentrate on surrogation requests arising from the hole within the reduction sequence $D[\overline{y}\langle \mathsf{op}^\star, q, j\rangle] \Rightarrow P \downarrow_p$ and call *significant* ($\rightarrow_{\mathrm{s}}$) precisely those steps that exhibit the commitment to an external $\mathsf{op}^\star$-request. The other steps—except for the cases of internal surrogation, which are precisely excluded by assumption—are *insignificant*: they can even be mimicked up to structural equivalence.

In order to make the proof work, we iterate the simulation steps along the given sequence $D[\overline{y}\langle \mathsf{sur}^\star_q, j\rangle] \Rightarrow P \downarrow_p$. Let us assume that this sequence has d significant steps and that we have iterated already $h-1$ of them, for $0 < h \leq d$:

$$D[\overline{y}\langle \mathsf{sur}^\star_q, j\rangle]\ (\Rightarrow \rightarrow_{\mathrm{s}})^{h-1}\ \boxed{P_{h-1}}\ \Rightarrow \rightarrow_{\mathrm{s}}\ \boxed{P_h} \equiv \widehat{E}[\ \mathsf{surO}^\star_{\mathbb{O}}\langle\, s_h, \ldots\rangle\ ]$$

By (the tagged counterparts of) Lemma 3, we can precisely localize the state of the committed object manager inside P_h after the significant step (c.f. § 4.2). With respect to the relabeling $\rho := [{}^{\mathsf{png}^\star}/_{\mathsf{sur}^\star}]$, by assumption and iteration, we also have the sequence:

$$D[\overline{y}\langle \mathsf{png}^\star_q, j\rangle]\ (\Rightarrow \rightarrow_{\mathrm{s}})^{h-1} \simeq_\Gamma\ \boxed{P_{h-1}\rho}\ \Rightarrow \rightarrow_{\mathrm{s}}\ \boxed{Q_h} \equiv \widehat{E}[\ \mathsf{pingO}^\star_{\mathbb{O}}\langle\, s_h, \ldots\rangle\ ]\rho$$

by consuming a $\mathsf{png}^\star$-request. Now, we apply (the tagged counterparts of) Theorem 1 and Lemma 2, and the fact that barbed equivalence $\simeq_\Gamma$ implies structural equivalence $\equiv$ and is preserved by relabeling and get $Q_h \simeq_\Gamma P_h\rho$. This means that we can mimic the significant steps, thus the whole sequence, up to $\simeq_\Gamma$.

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Monographs CS. Springer, 1996.

[2] R. M. Amadio, I. Castellani, and D. Sangiorgi. On Bisimulations for the Asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998.

[3] L. Cardelli. `obliq-std.exe` — Binaries for Windows NT. `http://www.luca.-demon.co.uk/Obliq/Obliq.html`, 1994.

[4] L. Cardelli. A Language with Distributed Scope. *Computing Systems*, 8(1):27–59, 1995. An exteded abstract as appeared in *Proceedings of POPL '95*

[5] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the join-calculus. In *Proceedings of POPL '96*, ACM, 1996.

[6] A. D. Gordon, P. D. Hankin, and S. B. Lassen. Compilation and Equivalence of Imperative Objects. In *Proceedings of FSTTCS '97*, *LNCS* 1346. Springer, 1997.

[7] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communication. In *Proceedings of ECOOP '91*, volume 512 of *LNCS*, Springer, 1991.

[8] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[9] H. Hüttel and J. Kleist. Objects as Mobile Processes. Research Series RS-96-38, BRICS, 1996. Presented at *MFPS '96*.
[10] J. Kleist and D. Sangiorgi. Imperative Objects and Mobile Processes. In *Proceedings of PROCOMET '98*. Chapman & Hall, 1998.
[11] M. Merro. *Local π: A Model for Concurrent and Distributed Programming Languages*. PhD thesis, Ecole des Mines, France, 2000.
[12] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In *Proceedings of ICALP '98*, volume 1443 of *LNCS*. Springer, 1998.
[13] R. Milner. The Polyadic π-calculus: A Tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.
[14] R. Milner and D. Sangiorgi. Barbed Bisimulation. In *Proceedings of ICALP '92*, volume 623 of *LNCS*. Springer, 1992.
[15] U. Nestmann. Mobile Objects (A Project Overview). In *Proceedings of FBT '99*. Herbert Utz Verlag, 1999.
[16] U. Nestmann, H. Hüttel, J. Kleist, and M. Merro. Aliasing Models for Mobile Objects. Accepted for *Journal of Information and Computation*. An extended abstract has appeared as Distinguished Paper in the *Proceedings of EUROPAR '99*, pages 1353–1368, LNCS 1685, September 1999, 1999.
[17] B. C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
[18] A. Philippou and D. Walker. On Transformations of Concurrent Object Programs. *Theoretical Computer Science*, 195(2):259–289, 1998.
[19] D. Sangiorgi. An Interpretation of Typed Objects into Typed π-calculus. *Information and Computation*, 143(1):34–73, 1998.
[20] D. Sangiorgi. Typed π-calculus at Work: A proof of Jones's parallelisation theorem on Concurrent Objects. *Theory and Practice of Object-Oriented Systems*, 5, 1999.
[21] C. L. Talcott. Obliq Semantics Notes. Unpublished note. Available from `clt@cs.stanford.edu`, Jan. 1996.
[22] D. Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

An Interpretation of Typed Concurrent Objects in the Blue Calculus

Silvano Dal Zilio

Microsoft Research, Cambridge, England

Abstract. We propose an interpretation of a typed concurrent calculus of objects based on the imperative object calculus of Abadi and Cardelli. The target of our interpretation is a version of the blue calculus, a variant of the π-calculus that directly contains functions, with record and first-order types. We show that reductions and type judgments are derivable in a rather simple and natural way, and that our encoding can be extended to recursive and self-types, as well as to synchronization primitives. We also use our encoding to prove some equational laws on objects.

1 Introduction

In the recent past, there has been a growing interest in the theoretical foundations of object-oriented and concurrent programming languages. One of the means used to explain object-oriented concepts, such as object types or self-referencing for example, has been to look for an interpretation of these concepts in simpler formalisms, such as typed λ-calculi. However, these interpretations are difficult, and very technical, due to the difficulties raised by the typing, and subtyping, of objects. To circumvent these problems, Abadi and Cardelli have defined a canonical object-oriented calculus, the ς-calculus [1], in which the notion of object is primitive, and they have developed and studied type systems for this calculus.

In this paper, we give a model of concurrent object computation based on a modeling of objects as processes. We introduce some derived notations for objects and we give their translation in a version of the blue calculus, $\boldsymbol{\pi}^\star$ [5], extended with records. We type Blue calculus processes using an implicit, first-order type system based on the simply typed λ-calculus.

Using these derived constructs, we give an interpretation of a concurrent and imperative version of ς defined by Gordon and Hankin, **concς** [13]. We prove that this interpretation preserves reduction, typing and subtyping judgments. Therefore, our encoding gives an interpretation of complex notions, such as method update or object types, in terms of more basic notions such as records, field selection and functional types. Consequently, we obtain a type-safe way to implement higher-order concurrent objects in the Blue calculus, and therefore in the π-calculus ($\boldsymbol{\pi}$). Moreover, we can validate possible extensions of **concς** and, what is more original, we can use the embedding of **concς** in the Blue calculus to do equational reasoning on the source calculus. As an example, we sketch the proof of an equational law between objects at the end of this paper.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 409–424, 2000.
© Springer-Verlag Berlin Heidelberg 2000

We organize the rest of the paper as follows. The next section introduces the Blue calculus using very simple intuitions taken from the λ-calculus execution model. This is an occasion to give an informal and intuitive presentation of the Blue calculus to the reader. Section 3 briefly introduces Gordon and Hankin's calculus of objects and gives its interpretation in $\boldsymbol{\pi}^\star$. We prove that **concς** is embedded in $\boldsymbol{\pi}^\star$ and that objects can be viewed as a particular kind of (linearly managed) resource. Section 4 is dedicated to the typing of processes and objects. It introduces a new type operator that is very well suited for typed continuation passing style transformations. Before concluding, we look at possible applications of our interpretation. Complete definition of the calculi and omitted proofs may be found in a long version of the paper [8].

2 The Blue Calculus

In the functional programming world, a program is ideally represented by a λ-calculus term, that is a term generated by the following grammar:

$$M, N ::= x \mid \lambda x.M \mid (MN)$$

We enrich this calculus with a set of constants: $a_1, a_2, \ldots$, called names, that can be interpreted as resource locations. We describe a very simple execution model for programs written in this syntax based on the notion of abstract machine (AM), and we enrich it until we obtain a model that exhibits concurrent behaviors similar to those expressible in the π-calculus. This abstract machine sets up the foundation of the Blue calculus that can therefore be viewed, at the same time, as a concurrent λ-calculus and as an applicative π-calculus.

An abstract machine is defined by a set of *configurations*, denoted $\mathcal{K}$, and a set of *transition rules*, $\mathcal{K} \to \mathcal{K}'$, which define elementary computing steps. In our setting, a machine configuration is a triple $\{\mathcal{E}; M; \mathcal{S}\}$ where $\mathcal{E}$ is a memory, or store, that is an association between names and programs; M is a program, that is a λ-term; $\mathcal{S}$ is a stack containing the arguments of functional calls. Initially an AM has an empty memory, denoted by the symbol ϵ, which can be extended with new *declarations* as in $(\mathcal{E} \mid \langle a_n = N \rangle)$. The stack has a similar structure and we use $(a_j, \mathcal{S})$ to denote the operation of adding the name a_j to the stack.

An execution of the functional AM starts in the initial configuration $\mathcal{K}_0$, with an empty stack and memory ($\mathcal{K}_0 \triangleq \{\epsilon; M; \epsilon\}$). The transition rules are defined as follows, where $M\{x \leftarrow a_j\}$ denotes the outcome of renaming all free occurrences of x in M to the name a_j.

$$\begin{aligned}
\{\mathcal{E}; a_j; \mathcal{S}\} &\to \{\mathcal{E}; N_j; \mathcal{S}\} && (\mathcal{E} = \cdots \mid \langle a_j \Leftarrow N_j \rangle \mid \cdots) \\
\{\mathcal{E}; \lambda x.M; (a_j, \mathcal{S})\} &\to \{\mathcal{E}; M\{x \leftarrow a_j\}; \mathcal{S}\} \\
\{\mathcal{E}; (MN); \mathcal{S}\} &\to \{(\mathcal{E} \mid \langle a_n = N \rangle); M; (a_n, \mathcal{S})\} && (a_n \text{ fresh name})
\end{aligned}$$

For example, to evaluate a function application we memorize the argument in a fresh memory location, and we add the name of this location to the stack.

At each computation step, the machine is in a configuration of the kind:

$$\mathcal{K}_n = \{(\langle \mathrm{a}_1 \ = \ N_1 \rangle \mid \cdots \mid \langle a_n \ = \ N_n \rangle); M; (a_{i_1}, \ldots, a_{i_k})\}$$

Where the indices $i_j \in 1..n$ for each $j \in 1..k$. Each configuration corresponds to a λ-term and, for example, $\mathcal{K}_n$ corresponds to $(Ma_{i_1} \ldots a_{i_k})\{a_1 \leftarrow N_1\} \ldots \{a_n \leftarrow N_n\}$. Therefore, to each extension of the functional AM corresponds a generalization of the λ-calculus. In this section, we improve the functional AM until we obtain an execution model that compares to that of $\boldsymbol{\pi}$. The calculus defined by the extended AM is the Blue calculus.

We start with simple syntactical modifications. We modify our notations to use a sequence of applications instead of a stack, recasting the standard configuration $\mathcal{K}_n$ into: $\langle a_1 \ = \ N_1 \rangle \mid \cdots \mid \langle a_n \ = \ N_n \rangle \mid (Ma_{i_1} \ldots a_{i_k})$. With these modifications, we can reformulate the transition rules in the following way, with the side condition that the name a is fresh in rule (χ):

$$\begin{array}{rll}
\langle a \ = \ N \rangle \mid \cdots \mid (aa_1 \ldots a_n) \ \rightarrow \ \langle a \ = \ N \rangle \mid \cdots \mid (Na_1 \ldots a_n) & & (\rho) \\
(\lambda x.M)a_1 \ldots a_n \ \rightarrow \ (M\{x \leftarrow a_1\})a_2 \ldots a_n & & (\beta) \\
(MN)a_1 \ldots a_n \ \rightarrow \ \langle a \ = \ N \rangle \mid Maa_1 \ldots a_n & & (\chi) \\
\mathcal{K} \rightarrow \mathcal{K}' \ \Rightarrow \ (\langle a \ = \ N \rangle \mid \mathcal{K}) \rightarrow (\langle a \ = \ N \rangle \mid \mathcal{K}') & & (\varpi)
\end{array}$$

In this new presentation, rule (β) corresponds to a simplified form of beta-reduction, where we substitute a name, and not a term, for a variable, whereas rule (ρ) can be interpreted as a form of communication. Nonetheless, whereas the classical π-calculus communication model is based on message synchronization, we use instead a particular kind of resource fetching.

A first improvement to the AM is to consider $\mid$ as an associative composition operator, and to allow multiple configurations in parallel. We do not choose a commutative operator. The idea is to separate in each configuration, the store from the active part, that is, to separate the memory from the evaluated term. We allow some commutations though, with the restriction that the evaluated term is always at the right of the topmost parallel composition. More formally, we consider the following structural rules for parallel composition, where $P \leftrightarrows Q$ means that both $P \rightarrow Q$ and $Q \rightarrow P$ holds.

$$(M_1 \mid M_2) \mid M_3 \leftrightarrows M_1 \mid (M_2 \mid M_3) \qquad (M_1 \mid M_2) \mid M_3 \leftrightarrows (M_2 \mid M_1) \mid M_3$$

As a result, we obtain an asymmetric parallel composition operator, like the one defined in **concς**, or the formal description of CML [10]. Another consequence is that we can replace rule (ρ) by the simpler rule:

$$\langle a \ = \ N \rangle \mid (aa_1 \ldots a_n) \ \rightarrow \ \langle a \ = \ N \rangle \mid (Na_1 \ldots a_n) \tag{2.1}$$

Roughly speaking, we have transformed our functional AM to a Chemical AM (CHAM) in the style of [4]. The most notable improvement is the possibility to

compose multiple configurations and, for example, to define configurations with multiple declarations for the same name. Indeed this introduces the possibility of non-deterministic transitions, such as:

$$(\langle a = N_1\rangle \mid \langle a = N_2\rangle \mid a) \to (\langle a = N_1\rangle \mid \langle a = N_2\rangle \mid N_1)$$
$$(\langle a = N_1\rangle \mid \langle a = N_2\rangle \mid a) \to (\langle a = N_1\rangle \mid \langle a = N_2\rangle \mid N_2)$$

Another improvement to our concurrent AM is the addition of a new kind of declaration, that is discarded after a communication. We denote $\langle a \Leftarrow P\rangle$ this declaration, and we add the following communication rule.

$$\langle a \Leftarrow N\rangle \mid (a a_1 \dots a_n) \to (N a_1 \dots a_n) \tag{2.2}$$

Intuitively, the declaration $\langle a \Leftarrow N\rangle$ allows us to control explicitly the number of accesses to the resource named a, like the input operator $a(x).P$ in $\boldsymbol{\pi}$, and thus it allows us to capture the evaluation blocking phenomena that are peculiar to concurrent executions. In the encoding of concurrent objects in $\boldsymbol{\pi}^\star$, we will see that objects also appear as a particular kind of declarations.

Finally, we add the possibility to dynamically create fresh names. This mechanism is a distinctive feature of the π-calculus, and it is very easily implemented in our CHAM by adding the restriction operator, $(\nu a)\mathcal{K}$, together with new reduction rules. Using the restriction operator we can, for example, define the internal choice operator $(M \oplus N)$ to be the term $(\nu a)(\langle a \Leftarrow M\rangle \mid \langle a \Leftarrow N\rangle \mid a)$.

2.1 The Calculus

The Blue calculus can be viewed as the calculus obtained from the concurrent AM, in the same way that the join-calculus is derived from the reflexive CHAM defined in [12]. The following table gives the syntax of processes, P. The syntax depends on a set of atomic names, $\mathcal{N}$, ranged over by $a, b, \dots$ and partitioned in three kinds: variables $x, y, \dots$, bound by abstractions, $(\lambda x)P$; references $u, v, \dots$, bound by restrictions, $(\nu u)P$; labels $k, l, \dots$, used to name record fields.

Processes

$P, Q ::=$	process
a	name
$(\lambda x)P$	small λ-abstraction
(Pa)	application
$(P \mid Q)$	parallel composition
$(\nu u)P$	name restriction
$\langle u \Leftarrow P\rangle$	linear declaration
$\langle u = P\rangle$	replicated declaration
$[\,]$	empty record
$[P, l = Q]$	record extension
$(P \cdot l)$	selection

Our syntax enforces a restricted usage of names with respect to their kinds: we only allow declaration on references and abstraction on variables. This rule out terms such as $(\lambda x)\langle x \Leftarrow P\rangle$ for example.

The formal operational semantics of $\boldsymbol{\pi}^\star$ is given in a chemical style and defined using two relations. First, the *structural congruence* relation $\equiv$, which is equivalent to the relation $\leftrightarrows$ used previously, and that is used to rearrange terms. Second, the *reduction* relation, $\rightarrow$, which represents real computation steps, and that corresponds to (2.1), (2.2) and (β).

Structural congruence

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\lambda x)P \equiv (\lambda x)Q$	(Struct Lam)
$P \equiv Q \Rightarrow (Pa) \equiv (Qa)$	(Struct Appl)
$P \equiv Q \Rightarrow (P \cdot l) \equiv (Q \cdot l)$	(Struct Sel)
$P \equiv Q \Rightarrow (P \mid R) \equiv (Q \mid R)$	(Struct Par)
$P \equiv Q \Rightarrow (\nu u)P \equiv (\nu u)Q$	(Struct Res)
$P \equiv Q \Rightarrow \langle u \Leftarrow P\rangle \equiv \langle u \Leftarrow Q\rangle$	(Struct Decl)
$P \equiv Q \Rightarrow \langle u = P\rangle \equiv \langle u = Q\rangle$	(Struct Mdecl)
$P \equiv Q, R \equiv S \Rightarrow [P, l = R] \equiv [Q, l = S]$	(Struct Over)
$((P \mid Q) \mid R) \equiv (P \mid (Q \mid R))$	(Struct Par Assoc)
$((P \mid Q) \mid R) \equiv ((Q \mid P) \mid R)$	(Struct Par Comm)
$u \notin \mathbf{fn}(Q) \Rightarrow (\nu u)P \mid Q \equiv (\nu u)(P \mid Q)$	(Struct Res Par)
$u \notin \mathbf{fn}(Q) \Rightarrow Q \mid (\nu u)P \equiv (\nu u)(Q \mid P)$	(Struct Par Res)
$(\nu u)(\nu v)P \equiv (\nu v)(\nu u)P$	(Struct Res Res)
$(P \mid Q)a \equiv P \mid (Qa)$	(Struct Par Appl)
$(P \mid Q) \cdot l \equiv P \mid (Q \cdot l)$	(Struct Par Sel)
$a \neq u \Rightarrow ((\nu u)P)a \equiv (\nu u)(Pa)$	(Struct Res Appl)
$((\nu u)P) \cdot l \equiv (\nu u)(P \cdot l)$	(Struct Res Sel)

Reduction

$P \rightarrow P' \Rightarrow (Pa) \rightarrow (P'a)$	(Red Appl)
$P \rightarrow P' \Rightarrow (P \cdot l) \rightarrow (P' \cdot l)$	(Red Sel)
$P \rightarrow P' \Rightarrow (P \mid Q) \rightarrow (P' \mid Q)$	(Red Par 1)
$Q \rightarrow Q' \Rightarrow (P \mid Q) \rightarrow (P \mid Q')$	(Red Par 2)
$P \rightarrow P' \Rightarrow (\nu u)P \rightarrow (\nu u)P'$	(Red Res)
$P \rightarrow P', P \equiv Q \Rightarrow Q \rightarrow P'$	(Red $\equiv$)
$((\lambda u)P)v \rightarrow P\{u \leftarrow v\}$	(Red Beta)
$\langle u \Leftarrow P\rangle \mid (ua_1 \ldots a_n) \rightarrow (Pa_1 \ldots a_n)$	(Red Decl)
$\langle u = P\rangle \mid (ua_1 \ldots a_n) \rightarrow \langle u = P\rangle \mid (Pa_1 \ldots a_n)$	(Red Mdecl)
$[P, l = Q] \cdot l \rightarrow Q$	(Red Sel)
$k \neq l \Rightarrow [P, l = Q] \cdot k \rightarrow P \cdot k$	(Red Over)

Notations. We use $\tilde{a}$ to denote the sequence $a_1, \ldots, a_n$ and $\mathbf{fn}(P)$ to denote the set of free names in P. We abbreviate a sequence of abstractions, $(\lambda x_1) \ldots (\lambda x_n) P$, into $(\lambda \tilde{x}) P$. The same convention applies for $(\nu \tilde{u}) P$. We also abbreviate a sequence of extensions, $[\,[\,[\,]\,, l_1 = P_1\,]\,, \ldots, l_n = P_n\,]$, where $l_1, \ldots, l_n$ are pairwise distinct labels, into $[\,l_i = P_i{}^{i \in 1..n}\,]$, and a sequence of applications, $P a_1 \ldots a_n$, into $P\tilde{a}$.

2.2 Derived Operators

To simplify the presentation of our encoding and of the type system, we introduce three derived operators.

$$
\begin{array}{rcll}
\mathbf{def}\, u = P \,\mathbf{in}\, Q & \triangleq & (\nu u)(\langle u = P \rangle \mid Q) & \text{definition} \\
\mathbf{set}\, x = P \,\mathbf{in}\, Q & \triangleq & (\nu u)(\langle u \Leftarrow (\lambda x) Q \rangle \mid (Pu)) & \text{linear application} \\
\mathbf{reply}(a) & \triangleq & (\lambda r)(r\ a) & \text{synchronous message}
\end{array}
$$

We may interpret the first operator, subsequently called a definition, as an explicit substitution of P for the name u in Q, and we can define higher-order application, (PQ), as a shorthand for $\mathbf{def}\, u = Q \,\mathbf{in}\, (Pu)$, where $u \notin \mathbf{fn}(P) \cup \mathbf{fn}(Q)$. Note that the name u is recursively bound in $\mathbf{def}\, u = P \,\mathbf{in}\, Q$, and that it is possible to define recursion, $\mathbf{rec}\, u.P$, by $(\mathbf{def}\, u = P \,\mathbf{in}\, u)$. Using linear application, it is possible to define sequential composition, $P\,;\,Q$, as $(\mathbf{set}\, u = P \ \mathbf{in}\, Q)$, for some u not free in Q. The **reply** operator is used in continuation passing style encoding and, for example, to return a value in a linear application.

$$
\mathbf{set}\, x = \mathbf{reply}(a) \,\mathbf{in}\, Q \ \rightarrow\ (\nu u)(\langle u \Leftarrow (\lambda x) Q \rangle \mid (ua)) \ \stackrel{*}{\rightarrow}\ Q\{x \leftarrow a\}
$$

We can compare $\mathbf{reply}(a)$ with the *synchronous names* of the join-calculus, with the difference that, using our notation for higher-order application, it is possible to "reply" a general term, with $\mathbf{reply}(P)$ standing for the term $\mathbf{def}\, u = P \,\mathbf{in}\, (\lambda r)(ru)$.

3 Interpretation of the Concurrent Object Calculus

The calculus **concς** is a calculus based on the notion of naming, obtained by extending the imperative ς-calculus with π-calculus primitives, such as parallel composition and restriction. As the imperative ς-calculus, it also provides an operator to clone an object, and a call-by-value definition operator: $\mathbf{let}\, x = a \,\mathbf{in}\, b$.

Expressions and results

$u, v ::=$	result
$\quad x$	variable
$\quad p$	name
$d ::=$	denotation
$\quad \{l_i = \varsigma(x_i) b_j{}^{i \in 1..n}\}$	

$a, b, c ::=$	term
u	result
$(p \mapsto d)$	denomination
$u{\cdot}l$	method invocation
$u{\cdot}l \Leftarrow \varsigma(x)b$	method update
$\mathbf{clone}(u)$	cloning
$\mathbf{let}\, x = a \,\mathbf{in}\, b$	sequencing
$(a \Rsh b)$	parallel composition
$(\nu p)a$	name restriction

The basic constructor of **concς** is the denomination, $(p \mapsto d)$, that, informally, adds a name to an object of **ς** and acts like a special kind of declaration. It represents the store of an object-oriented program, like the declaration $\langle a = M \rangle$ represents the store of a configuration in the functional AM. We omit the formal definition of **concς** in this paper, but we hope that the reader unfamiliar with this calculus can grasp some idea of it from our interpretation. Its operational semantics is defined in a chemical style, with a structural equivalence, $\equiv$, analogous to the homonymous relation in $\boldsymbol{\pi}^\star$, and a reduction relation, $\rightarrow$. Some reductions of **concς** come from the **let** constructor, where values are names. For example, $(\mathbf{let}\, x = p \,\mathbf{in}\, b) \rightarrow b\{x{\leftarrow}p\}$. However, the basic interactions are between a denomination, and a method invocation, a method update or a cloning on its name. Assume d is the denotation $\{l_i = \varsigma(x_i)b_i{}^{i\in 1..n}\}$, we get that:

$$\frac{j \in 1..n}{(p \mapsto d) \Rsh p{\cdot}l_j \rightarrow (p \mapsto d) \Rsh b_j\{x_j{\leftarrow}p\}}$$

$$\frac{d' \triangleq \{l_i = \varsigma(x_i)b_i{}^{i\in 1..n, i\neq j}, l_j = \varsigma(x)b\} \qquad j \in 1..n}{(p \mapsto d) \Rsh (p{\cdot}l_j \Leftarrow \varsigma(x)b) \rightarrow (p \mapsto d') \Rsh p}$$

$$\frac{q \notin \mathbf{fn}(d)}{(p \mapsto d) \Rsh \mathbf{clone}(p) \rightarrow (p \mapsto d) \Rsh (\nu q)((q \mapsto d) \Rsh q)}$$

We interpret **concς** in the Blue calculus and we prove an operational correspondence result. In the process of defining the encoding of **concς**, denoted $[\![.]\!]$ hereafter, we will naturally introduce some derived operators for the object notations. We see that it allows regarding **concς** as embedded in the Blue calculus, and therefore $\boldsymbol{\pi}^\star$ as an object calculus.

We suppose that the **concς** names are included in $\boldsymbol{\pi}^\star$. Informally, the interpretation of a denomination $(p \mapsto d)$, where $d \triangleq \{l_i = \varsigma(x_i)b_i{}^{i\in 1..n}\}$, is a process modeling a "reference cell" that memorizes n values, $(\lambda x_1)[\![b_1]\!]$, ... , $(\lambda x_n)[\![b_n]\!]$. That is a recursively defined declaration of the name p, which encapsulates a record with $2n$ fields: the access field get_{l_i}, used to invoke method l_i; the field put_{l_i}, used to modify this method. We also add a field named *clone* that, when selected, creates a fresh cell with a copy of the current state. Schematically, we use the *split-method* technique of [2].

Let $\mathbf{R}(p, s, \tilde{x}, c)$ be the following record (of $\boldsymbol{\pi}^\star$):

$$\mathbf{R}(p, s, \tilde{x}, c) \triangleq \left[\begin{array}{l} \ldots \\ get_{l_i} = (s\tilde{x} \mid x_i p), \\ put_{l_i} = (\lambda y)(s x_1 \ldots x_{i-1}\, y\, x_{i+1} \ldots x_n \mid \mathbf{reply}(p)), \\ \ldots \\ clone = (s\tilde{x} \mid c\tilde{x}) \end{array}\right]^{i \in 1..n}$$

In our intuition, the identity of an object is a reference at which the object state can be fetched (the name p), its state is a record of methods as in the classical recursive records semantics [6] and encapsulation is naturally implemented using the **def** operator. In particular, the variable x_i is used to record the value of the method l_i, the name s is a pointer to a function that (given the x_i's) creates the object each time it is accessed, and the name c is a pointer to a function that creates a fresh copy of the object when it is cloned.

To encode a denomination, we encapsulate the record $\mathbf{R}(p, s, \tilde{x}, c)$ in a recursive definition that linearly manages a declaration of the name p. We define a notation for this definition.

$$\mathbf{Fobj}(p, \tilde{x}, c) \triangleq \mathbf{def}\ s = (\lambda \tilde{y})\langle p \Leftarrow \mathbf{R}(p, s, \tilde{y}, c)\rangle\ \mathbf{in}\ \langle p \Leftarrow \mathbf{R}(p, s, \tilde{x}, c)\rangle$$

We denote $\langle p \hookleftarrow \{\, l_i = (\lambda x_i) P_i^{\,i \in 1..n} \,\}\rangle$ the process that we obtain by binding the name c to the function that clones the object, and the names in $\tilde{x}$ to the functions $(\lambda x_1)P_1, \ldots, (\lambda x_n)P_n$.

$$\langle p \hookleftarrow \{l_i = (\lambda x_i) P_i^{\,i \in 1..n}\}\rangle \triangleq \left(\begin{array}{l} \mathbf{def}\ c = (\lambda \tilde{x})(\nu q)(\mathbf{Fobj}(q, \tilde{x}, c) \mid \mathbf{reply}(q)) \\ \mathbf{in\ def}\ u_1 = (\lambda x_1)P_1, \ldots, u_n = (\lambda x_n)P_n \\ \quad \mathbf{in}\ \mathbf{Fobj}(p, \tilde{u}, c) \end{array}\right)$$

Intuitively, the process $\langle p \hookleftarrow D\rangle$, where $D \triangleq \{l_i = (\lambda x_i) P_i^{\,i \in 1..n}\}$, can be divided into two components. An *active part*, the declaration $\langle p \Leftarrow \mathbf{R}(p, s, \tilde{x}, c)\rangle$, which can interact with other processes in parallel. A *passive part*, the recursive definitions on the names s, $\tilde{x}$ and c, which are used to memorize the internal state of the cell and to (linearly) recreate its active part each time the name p is invoked. Indeed, when $\langle p \hookleftarrow D\rangle$ interacts with the name p, the unique declaration on p is consumed and a unique output on the restricted name s, acting like a lock, is freed, which, in turn, frees a single declaration on p. Using the derived operator $\langle p \hookleftarrow D\rangle$, we give a very simple and direct interpretation of **concς**.

Translation rules

$[\![(p \mapsto \{l_i = \varsigma(x_i) b_i^{\,i \in 1..n}\})]\!] \triangleq \langle p \hookleftarrow \{\, l_i = (\lambda x_i)[\![b_i]\!]^{\,i \in 1..n} \,\}\rangle$
$[\![u]\!] \triangleq \mathbf{reply}(u)$
$[\![u{\cdot}l]\!] \triangleq (u \cdot get_l)$
$[\![u{\cdot}l \Leftarrow \varsigma(x) b]\!] \triangleq (u \cdot put_l\, (\lambda x)[\![b]\!])$

$[\![\mathbf{clone}(u)]\!] \triangleq (u \cdot clone)$
$[\![\mathbf{let}\, x = a \,\mathbf{in}\, b]\!] \triangleq (\mathbf{set}\, x = [\![a]\!] \,\mathbf{in}\, [\![b]\!])$
$[\![a \Rsh b]\!] \triangleq ([\![a]\!] \mid [\![b]\!])$
$[\![(\nu p)a]\!] \triangleq (\nu p)[\![a]\!]$

We can simplify this interpretation a step further by defining three shorthand for method select, method update and for cloning.

$$(P \Leftarrow l) \triangleq (P \cdot get_l) \quad (P \cdot l \Leftarrow (\lambda x)Q) \triangleq (P \cdot put_l\ (\lambda x)Q) \quad \mathbf{clone}(P) \triangleq (P \cdot clone)$$

With these notations we can consider that **concς** is directly embedded in the Blue calculus. More interestingly, we embed a higher-order version of the object calculus, and it is possible to define terms that are not in **concς**, like $\mathbf{clone}(P \mid Q)$ for example, or the selector function $(\lambda x)(x \Leftarrow l)$. We can also derive a set of reduction sequences that simulate reduction in **concς**. Assume D is the association $\{\, l_i = (\lambda x_i)P_i^{\,i \in 1..n} \,\}$ and $j \in 1..n$ then:

$$\langle p \leftarrowtail D \rangle \mid p \Leftarrow l_j \;\xrightarrow{*}\; \langle p \leftarrowtail D \rangle \mid P_j\{x_j \leftarrow p\}$$

More formally, we prove that there is an operational correspondence between **concς** and the Blue calculus. To state this result, we use an observational equivalence between $\boldsymbol{\pi}^\star$-terms defined in [7], denoted $\approx$, that is a variant of weak barbed congruence [20]. Informally, this relation is the largest bisimulation that preserves simple observations called barbs and that is a congruence.

Theorem 3.1. *If $a \equiv b$, then $[\![a]\!] \equiv [\![b]\!]$. If $a \to a'$, then $[\![a]\!] \xrightarrow{*} \approx_b [\![a']\!]$. If $[\![a]\!] \to P$, then there exists a **concς**-term, a', such that $a \to a'$ and $P \xrightarrow{*} \approx_b [\![a']\!]$.*

4 Type System

We define a first-order type system for $\boldsymbol{\pi}^\star$, inspired by the (Curry-style) simply typed λ-calculus. It is essentially the type system given in [5], extended with subtyping, record types, recursion and a special type constructor for continuations, *Reply*(.). Then, we establish a set of derived typing rules for the object notations introduced in the previous section, that simulate the typing rules of **concς**.

We assume the existence of a denumerable set of type variables ranged over by $\alpha, \beta, \ldots$ The syntax of type expressions is given by the following grammar.

$$\begin{array}{lll} \tau, \vartheta, \varrho \;::=\; \alpha \mid (\tau \to \vartheta) \mid (\mu\alpha.\tau) \mid \mathit{Top} \mid & & \text{simple types} \\ \qquad [\,] \mid [\,\varrho\,,\, l : \tau\,] \mid \mathit{Reply}(\tau) & & \text{rows \& continuation type} \end{array}$$

We consider that types are well formed with respect to a simple kinding system, described in the extended version of this paper [8]. Informally, the kind system is used to constrain the type ϱ in the row $[\,\varrho\,,\, l : \tau\,]$ and, for example, to rule out types such as $[\,(\vartheta \to \varrho)\,,\, l : \tau\,]$.

In our system, a type environment is an association between names and types, and also between type variables and kinds: $\Gamma ::= \emptyset \mid \Gamma, a : \tau \mid \Gamma, \alpha :: \kappa$. The type system is based on four judgments: (1) $\Gamma \vdash \diamond$, and (2) $\Gamma \vdash \tau :: \kappa$, for well formed environment and types; (3) $\Gamma \vdash \tau <: \vartheta$, and (4) $\Gamma \vdash P : \tau$, given Γ, type τ is a subtype of ϑ and term P has type τ.

The type constructors are all borrowed from type systems for functional languages, apart from *Reply*(.) that is used to type the continuation operator **reply**(P) (and linear application) and that is described later. The type *Top* is the maximal type with respect to the subtyping relation. We make a non-standard use of this type constant: *Top* is used to type terms that may not be expected to return results, for example resources.

$$\frac{\Gamma \vdash P : \tau \qquad (u : \tau) \in \Gamma}{\Gamma \vdash \langle u \Leftarrow P \rangle : \mathit{Top}} \text{ (Proc Decl)} \qquad \frac{\Gamma \vdash P : \mathit{Top} \qquad \Gamma \vdash Q : \tau}{\Gamma \vdash (P \mid Q) : \tau} \text{ (Proc Par)}$$

Subtyping. We do not give the details of the subtyping rules here. The rules for the functional part of the system are standard. For example arrow types $(\tau \to \vartheta)$ are *contravariant* in the first parameter and *covariant* in the second. The subtyping rules for rows are less classical, and reflect the incremental construction of records. Provided the rows are well formed, we have the following subtyping rules, together with rules that allow identifying rows up-to reordering of their components.

$$\frac{}{\Gamma \vdash [\,\varrho\,,\, l : \tau\,] <: [\,]} \qquad \frac{\Gamma \vdash \varrho <: \varrho' \qquad \Gamma \vdash \tau <: \tau'}{\Gamma \vdash [\,\varrho\,,\, l : \tau\,] <: [\,\varrho'\,,\, l : \tau'\,]}$$

Typing rules. The typing rules for the functional part of the calculus are those of the simply typed λ-calculus extended with records and subtyping. The typing rules for the π-calculus operators are the rules (Proc Par) and (Proc Decl) defined previously. In particular, the type of a parallel composition, $P \mid Q$, is the type of the main thread of computation, which is Q. The typing rule for declarations, (Proc Decl), deserves more comment. Suppose that $\Gamma \vdash P : \vartheta$, with $(u : \tau) \in \Gamma$, and that u appears in subject position of a declaration $\langle u \Leftarrow Q \rangle$, for example $P \triangleq (\langle u \Leftarrow Q \rangle \mid R)$. Since we may substitute Q for an occurrence of u in R, see (2.2), the term Q must have the type τ. Using rule (Proc Decl), it is easy to derive typing rules for definitions and higher-order application, that are equivalent to those found in the ML type system.

$$\frac{\Gamma, u : \tau \vdash P : \tau \qquad \Gamma, u : \tau \vdash Q : \sigma}{\Gamma \vdash \mathbf{def}\, u = P \,\mathbf{in}\, Q : \sigma} \qquad \frac{\Gamma \vdash P : \tau \to \vartheta \qquad \Gamma \vdash Q : \tau}{\Gamma \vdash (P\,Q) : \vartheta}$$

Typing continuations. We explain the typing and subtyping rules for the operator *Reply*(.). Recall that **reply**(a) stands for $(\lambda r)(ra)$. Let α be a fresh type variable. It can be proved that if P has type τ, then $(\lambda r)(r\,P)$ has type $(\tau \to \alpha) \to \alpha$. In $(\tau \to \alpha) \to \alpha$, the variable α is implicitly quantified, in the sense that **reply**(P) can be given the type $\forall\alpha.((\tau \to \alpha) \to \alpha)$ in the ML type system. The type $(\tau \to \alpha) \to \alpha$ is often found in typed continuation passing style transformations, and in

λ-calculi with exceptions, where it is sometimes denoted $\neg^{\alpha}\neg^{\alpha}\tau$ [15]. To avoid the introduction of quantified types, we use a new operator to type the term $\mathbf{reply}(P)$, together with the introduction rule:

$$\frac{\Gamma \vdash P : \tau}{\Gamma \vdash \mathbf{reply}(P) : \mathit{Reply}(\tau)}$$

We can compare our usage of $\mathbf{reply}(.)$ with the usage of the operator **let** in ML, that can be defined as syntactic sugar for the term $(\lambda x.M)\,N$, but that is used in the type system to introduce parametric polymorphism.

It is possible to validate the two following rules using the interpretation of $\mathit{Reply}(\tau)$ as the type $(\tau \to \alpha) \to \alpha$ (for some fresh type variable α).

$$\frac{\Gamma \vdash P : \mathit{Reply}(\tau) \qquad \Gamma, x : \tau \vdash Q : \vartheta}{\Gamma \vdash \mathbf{set}\, x = P\ \mathbf{in}\, Q : \vartheta} \qquad \frac{\Gamma \vdash \tau <: \vartheta}{\Gamma \vdash \mathit{Reply}(\tau) <: \mathit{Reply}(\vartheta)}$$

The presence of $\mathit{Reply}(.)$ is only a minor extension to the traditional type system of $\boldsymbol{\pi}^{\star}$, and it does not modify its interesting properties. In particular, we prove that reduction preserves type judgments.

Theorem 4.1. *If $\Gamma \vdash P : \tau$ and $P \to P'$, then $\Gamma \vdash P' : \tau$.*

Typing objects. We prove a typed correspondence between **concς** and the Blue calculus. To simplify our presentation, we first define a special notation for the type of a denomination. Apart from the use of the operator $\mathit{Reply}(.)$, this type is analogous to the one obtained in the encoding of Abadi-Cardelli functional object calculus given by Viswanathan [25] and Sangiorgi [23].

$$\mathrm{Obj}(\alpha.[\, l_i : \vartheta_i{}^{i\in 1..n}\,]) \triangleq \mu\alpha. \begin{bmatrix} \ldots & {\scriptstyle i\in 1..n} \\ \mathit{get}_{l_i} : \vartheta_i, & \\ \mathit{put}_{l_i} : (\alpha \to \vartheta_i) \to \mathit{Reply}(\alpha), & \\ \ldots & \\ \mathit{clone} : \mathit{Reply}(\alpha) & \end{bmatrix}$$

We prove that the object type, $\mathrm{Obj}(\alpha.\varrho)$, is the type of the name p in $\langle p \leftarrow\!\!\!\rightarrow D\rangle$. In $\mathrm{Obj}(\alpha.\varrho)$, the variable α is called the *self-type*. We simply write $\mathrm{Obj}(\varrho)$ if the self-type does not appear free in ϱ. We can give derived typing rules for the objects constructs defined in Sect. 3.

Derived typing rules for the embedding of objects

Assume A is the type $\mathrm{Obj}(\alpha.[\, l_i : \vartheta_i^{i\in 1..n}\,])$.

$$\frac{(p : \mathrm{A}) \in \Gamma \qquad \forall i \in 1..n \qquad \Gamma, x_i : \mathrm{A} \vdash P_i : \vartheta_i\{\alpha{\leftarrow}\mathrm{A}\}}{\Gamma \vdash \langle p \leftarrow\!\!\!\rightarrow \{\, l_i = (\lambda x_i) P_i{}^{i\in 1..n}\,\}\rangle : \mathit{Top}} \text{ (Proc Obj)}$$

$$\frac{\Gamma \vdash P : \mathrm{A} \qquad j \in 1..n \qquad \Gamma, x : \mathrm{A} \vdash Q : \vartheta_j\{\alpha{\leftarrow}\mathrm{A}\}}{\Gamma \vdash (P{\cdot}l_j \Leftarrow (\lambda x)Q) : \mathit{Reply}(\mathrm{A})} \text{ (Proc Updt)}$$

$$\frac{\Gamma \vdash P : \mathrm{A}}{\Gamma \vdash \mathbf{clone}(P) : \mathit{Reply}(\mathrm{A})} \text{ (Proc Clone)} \qquad \frac{\Gamma \vdash P : \mathrm{A} \qquad j \in 1..n}{\Gamma \vdash P \Leftarrow l_j : \vartheta_j\{\alpha{\leftarrow}\mathrm{A}\}} \text{ (Proc Invk)}$$

We give only the derivation for method update. Recall that $(P{\cdot}l_j \Leftarrow (\lambda x)Q)$ denotes the term $(P \cdot put_{l_j}\,(\lambda x)Q)$. Let A denotes the type $\mathrm{Obj}(\alpha.[\,l_i : \vartheta_i{}^{i\in 1..n}\,])$. Suppose that j is in $1..n$, that $\Gamma \vdash P : A$, and that $\Gamma, x : A \vdash Q : \vartheta_j\{\alpha{\leftarrow}A\}$. It follows that $\Gamma \vdash (\lambda x)Q : (\alpha \to \vartheta_j)\{\alpha{\leftarrow}A\}$, and that $\Gamma \vdash (P \cdot put_{l_j}) : ((\alpha \to \vartheta_j) \to \mathit{Reply}(\alpha))\{\alpha{\leftarrow}A\}$. Hence $\Gamma \vdash (P{\cdot}l_j \Leftarrow (\lambda x)Q) : \mathit{Reply}(A)$. Note that it is impossible to extend the object P with a new method, since the set $(put_{l_j})_{j\in 1..n}$ of fields is fixed. Moreover, like in Abadi-Cardelli calculus of first-order objects, it is impossible to refine the type of the updated method. Indeed, the type ϑ_j appears in contravariant position in field put_{l_j}, and in covariant position in field get_{l_j}. For the same reason, we can prove that object types are not covariant, that is $\varrho <: \sigma$ does not imply $\mathrm{Obj}(\varrho) <: \mathrm{Obj}(\sigma)$. However, we prove that width subtyping between object interfaces is sound.

Lemma 4.1. $\mathrm{Obj}([\,l_i : \vartheta_i{}^{i\in 1..n+m}\,]) <: \mathrm{Obj}([\,l_i : \vartheta_i{}^{i\in 1..n}\,])$.

The type system of **concς** is based on Abadi-Cardelli first-order object calculus, $\mathbf{Ob}_{1<:}$, extended with new type constants for expressions, processes and synchronization. In this system, a clear distinction is made between *expressions*, that is terms expected to return results, and *processes*, that intuitively represent stores of expressions. Then, the type system is used for two different goals. First, to guarantee that terms are well formed and that a name cannot be associated to two different denominations. Second, to avoid runtime errors, which are instances of the so-called "message not understood" problem. In this paper, we study a version that only guarantee safety of executions, but it is not difficult to extend our type system to accommodate the first requirement, as in type system ensuring the "unique receiver" property in $\boldsymbol{\pi}$ [3].

As in $\mathbf{Ob}_{1<:}$, the basic type constructor is $[\,l_i : A_i{}^{i\in 1..n}\,]$, the type of objects with methods $l_i{}^{i\in 1..n}$, returning results of types $A_i{}^{i\in 1..n}$ respectively. There is also a constant, *Proc*, used to type processes, like denominations for example. The type system is based on a subtyping relation, $E \vdash A <: B$, such that *Proc* is the maximal type and that $[\,l_i : A_i{}^{i\in 1..n+m}\,] <: [\,l_i : A_i{}^{i\in 1..n}\,]$.

$[\![\,[\,l_i : A_i{}^{i\in 1..n}\,]\,]\!] \triangleq \mathrm{Obj}([\,l_i : \mathit{Reply}([\![A_i]\!])^{i\in 1..n}\,])$	$[\![\mathit{Proc}]\!] \triangleq \mathit{Top}$
$[\![E, x : A]\!] \triangleq [\![E]\!], x : [\![A]\!]$	$[\![\emptyset]\!] \triangleq \emptyset$

Theorem 4.2. *The interpretation preserves subtyping judgments: if* $E \vdash A{<:}B$, *then* $[\![E]\!] \vdash [\![A]\!] <: [\![B]\!]$. *The interpretation preserves typing judgments: if* $E \vdash a : \mathit{Proc}$, *then* $[\![E]\!] \vdash [\![a]\!] : \mathit{Top}$. *If* $E \vdash a : A$ *and* $A \neq \mathit{Proc}$, *then* $[\![E]\!] \vdash [\![a]\!] : \mathit{Reply}([\![A]\!])$.

In fact, we can prove a more general result than Theorem 4.2 since the type system for **concς** does not have recursive types, or self types, while our interpretation can capture such notions.

There is another modification to **concς** inspired by our interpretation. It consists in separating the two distinct roles of process and maximal type, that is to consider two different constants, *Top* and *Proc*, such that *Top* is the maximal

type and that *Proc* is used to type denominations. These two roles are collapsed in **concς**, as well as in the variant of $\boldsymbol{\pi}^\star$ defined in this paper. This emphasizes the fact that, in a parallel composition $(a \Rsh b)$, the value of a can never be communicated to the outside world, and thus only its side effects are observable. The fact that the value returned by a term is lost is not an example of a "runtime error", but we can consider that it is a programming mistake and, with our proposed modification, we can statically catch these mistakes.

5 Two Applications of our Interpretation

A first application is the interpretation of synchronization primitives. Although **concς** is a concurrent calculus, in the sense that multiple threads of computation can interact in parallel, it is not obvious how to synchronize these threads. The approach taken in [13] is to extend **concς** with operators for mutexes, that are defined as special kinds of denominations. The Blue calculus has a natural notion of synchronization based on asynchronous communication, exactly like in $\boldsymbol{\pi}$. Therefore, it is not surprising that our interpretation can be easily extended to model mutexes. What is more interesting is that our interpretation is also sound with respect to the typing rules for mutexes given in [13], and that mutexes appear (again) as a special kind of linearly defined resources.

A second application, that is the most original part of this work, is to prove equational laws between objects using barbed congruence between $\boldsymbol{\pi}^\star$-terms and our encoding. Let $\approx$ be the weak barbed congruence relation used in Theorem 3.1. We can use our interpretation to prove that two **concς**-terms are equivalent, by showing that their translations are equivalent. For example, if p is not free in d, we prove the following rules (among others):

$$[\![(\nu p)((p \mapsto d) \Rsh \mathbf{clone}(p))]\!] \approx [\![(\nu p)((p \mapsto d) \Rsh p)]\!]$$

$$\left[\!\!\left[(\nu p)\left(\begin{array}{l}(p \mapsto d)\Rsh\\ \mathbf{let}\, x = (p{\cdot}l \Leftarrow \varsigma(y)b)\,\mathbf{in}\, x{\cdot}l\end{array}\right)\right]\!\!\right] \approx \left[\!\!\left[(\nu p)\left(\begin{array}{l}(p \mapsto d)\Rsh\\ \mathbf{let}\, y = (p{\cdot}l \Leftarrow \varsigma(y)b)\,\mathbf{in}\, b\{y{\leftarrow}p\}\end{array}\right)\right]\!\!\right]$$

The first rule can be viewed as a concurrent version of an equational law proved for the imperative ς-calculus in [14], namely $(\mathbf{let}\, x = o\,\mathbf{in}\,\mathbf{clone}(x)) \approx o$, where o is the object $(\nu p)((p \mapsto d) \Rsh p)$, and p is not free in d.

An interesting fact is that the proofs of these equalities are very simple. Indeed, we only need to use "well-known" algebraic laws already proved for $\boldsymbol{\pi}^\star$ [7], like relation (5.1) below, similar to the replication theorem found in [19].

$$\mathbf{def}\, x = R\,\mathbf{in}\,(P \mid Q) \;\approx\; (\mathbf{def}\, x = R\,\mathbf{in}\,P) \mid (\mathbf{def}\, x = R\,\mathbf{in}\,Q) \qquad (5.1)$$

Another interesting fact is that the algebraic laws obtained on **concς** are still valid if one extends this calculus with new primitives that can be encoded in $\boldsymbol{\pi}^\star$, such as mutexes for example. Therefore, it is not necessary to modify the proof system each time the object calculus is extended.

As an example, we sketch the proof of the first equality. We use the notation of Sect. 3. Let $\mathbf{E_p}[.]$ be the context such that $[\![(p \mapsto d)]\!] = \mathbf{E_p}[\langle p \Leftarrow \mathbf{R}(p, s, \tilde{u}, c)\rangle]$.

$$
\begin{aligned}
[\![(\nu p)((p \mapsto d) \Rsh \mathbf{clone}(p))]\!] &\equiv (\nu p)\mathbf{E_p}[\langle p \Leftarrow \mathbf{R}(p, s, \tilde{u}, c)\rangle \mid p \cdot clone] && (1)\\
&\approx (\nu p)\mathbf{E_p}[\mathbf{R}(p, s, \tilde{u}, c) \cdot clone] && (2)\\
&\approx (\nu p)\mathbf{E_p}[s\tilde{u} \mid c\tilde{u}] && (3)\\
&\approx (\nu p)\mathbf{E_p}[s\tilde{u} \mid ((\lambda\tilde{x})(\nu q)(\mathbf{Fobj}(q, \tilde{x}, c) \mid \mathbf{reply}(q)))\tilde{u}] && (4)\\
&\approx (\nu p)\mathbf{E_p}[s\tilde{u} \mid (\nu q)(\mathbf{Fobj}(q, \tilde{u}, c) \mid \mathbf{reply}(q))] && (5)\\
&\approx (\nu p)(\mathbf{E_p}[s\tilde{u}] \mid (\nu q)([\![(q \mapsto d)]\!] \mid \mathbf{reply}(q))) && (6)\\
&\approx (\nu p)([\![(p \mapsto d)]\!] \mid (\nu q)([\![(q \mapsto d)]\!] \mid \mathbf{reply}(q))) && (7)\\
&\approx (\nu q)([\![(q \mapsto d)]\!] \mid [\![q]\!]) && (8)
\end{aligned}
$$

Step (1) uses an instance of the law: $(\nu u)(\langle u \Leftarrow P\rangle \mid u) \approx (\nu u)P$, and step (3) uses an instance of: $(\mathbf{def}\, x = R \,\mathbf{in}\, x) \approx (\mathbf{def}\, x = R \,\mathbf{in}\, R)$. In step (2) and (4), we use the fact that selection and β-reduction are deterministic reduction steps. For example we prove that $((\lambda x)P)a \approx P\{x{\leftarrow}a\}$. In step (5), we use (5.1) to distribute the definitions of $\mathbf{E_p}[.]$ over parallel composition, and in step (6) we use an intermediary result: $(\nu p)\mathbf{E_p}[s\tilde{u}] \approx (\nu p)[\![(p \mapsto d)]\!]$, that is implied by the laws used in step (3) and (4). In step (7), we use a "garbage collection" law similar to the following law: $((\nu u)\langle u \Leftarrow P\rangle) \mid Q \approx Q$.

6 Conclusion and Related Work

We have shown how to derive reduction and type judgments of **concς** in the Blue calculus in a rather simple and natural way. In our encoding, we model objects as a particular kind of declarations, $\langle p \leftarrowtail D\rangle$, that are "linearly managed". It is interesting to compare these declarations with the consumable declarations, $\langle u \Leftarrow P\rangle$, used to model processes (of the π-calculus), and with the replicated and immutable declarations, $\langle u = P\rangle$, used to model functions (of the λ-calculus).

Many theoretical studies address the problem of modeling object-oriented languages in procedural languages, but few of them have succeeded to preserve powerful features such as subtyping. In [2], the authors propose a compositional interpretation of a typed (sequential) object calculus with subtyping into $\mathtt{F}_{\leq\mu}$, a λ-calculus with second-order polymorphic types. Viswanathan improved this result in [25], where he gives a fully abstract interpretation in a first-order λ-calculus with reference cells and records. In both solutions, the encoding relies on the so-called split method. Fisher and Mitchell [11] proposed another interesting typed object calculus. However, none of those calculi can model concurrent and interactive objects.

In [23], Sangiorgi gives the first interpretation of Abadi-Cardelli typed functional calculus with subtyping in π (see also [17]). This interpretation is extended to the imperative case in [18, 21]. These interpretations, and the type system used, are very different from ours. For example, in the coding of method

update, we do not use *relay constructs*. Intuitively, in our encoding, the number of reductions when invoking a method does not depend on the number of method updates applied on the object. Therefore, if these encoding were used to implement concurrent objects, we would provide a more efficient implementation of method invocation. Another major difference is that, in the proof of the operational correctness property, we do not use a typed bisimulation.

There are also other formalisms used to model concurrent objects, mainly based on the π-calculus, such as [9, 12, 16, 22, 24], that are not considered in this paper.

We can compare our work with the proposal of [25], where the author gives a syntax-oriented interpretation of a typed object calculus. Our approach brings the same benefits as his. In particular, our interpretation defines a type-safe way of implementing higher-order concurrent objects in the Blue calculus, and therefore in $\boldsymbol{\pi}$.

A benefit of our encoding is that we validate some possible extensions of **concς**, like the extension of the type system with recursive types and self-types, or the extension with a maximal types, say *Top*, that differs from the type given to processes. Another interesting extension considered in this paper is the addition of functions and higher-order constructs to **concς**. Indeed, functions can be coded in Abadi-Cardelli object calculus, but to simulate the types of functions in a satisfactory way, they need to use universally and existentially quantified types to the detriment of type inference [1]. With our approach, we propose a natural extension of the object calculus with functions without noticeably modifying the definition of the equivalence or the type system, nor the interesting equational laws. Another benefit is the study of equational laws between objects. We give an example of such equational laws at the end of Section 5. It would be interesting to study the equivalence obtained on **concς** using barbed congruence and our encoding.

Acknowledgments. This work took place in the context of collaboration with Gérard Boudol at INRIA Sophia-Antipolis. He has greatly influenced the present development. I had useful converstions with Andrew Gordon, Paul Hankin, Massimo Merro and Davide Sangiorgi.

References

1. M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.
2. M. Abadi, L. Cardelli, and R. Viswanathan. An interpretation of objects and object types. In *Proc. of POPL '96*, pages 396–409, 1996.
3. R. Amadio and S. Prasad. Localities and failures. In *Proc. of FST & TCS '94*, volume 880 of *SLNCS*, pages 205–216, 1994.
4. G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
5. G. Boudol. The π-calculus in direct style. *Higher-Order and Symbolic Computation*, 11:177–208, 1998. Also appeared in *Proc. of POPL '97*, Jan. 1997.

6. L. Cardelli and J. C. Mitchell. Operations on records. *Math. Structures in Computer Science*, 1(1):3–48, 1991.
7. S. Dal-Zilio. A bisimulation for the Blue calculus. TR 3664, INRIA, Apr. 1999.
8. S. Dal-Zilio. An interpretation of typed concurrent objects in the Blue calculus. Extended version, available at `http://research.microsoft.com/~sdal/`, 1999.
9. P. Di Blasio and K. Fisher. A calculus for concurrent objects. In *Proc. of CONCUR '96*, volume 1119 of *LNCS*, Aug. 1996.
10. W. Ferreira, M. Hennessy, and A. Jeffrey. Combining typed λ-calculus with CCS. In *Essays in Honour of Robin Milner*. MIT Press, 1998.
11. K. Fisher and J. C. Mitchell. A delegation-based object calculus with subtyping. In *Proc. of FCT '95*, volume 965 of *LNCS*, pages 43–61, 1995.
12. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proc. of POPL '96*, pages 372–385, Jan. 1996.
13. A. D. Gordon and P. D. Hankin. A concurrent object calculus: reduction and typing. In *Proc. of HLCL '98*, Elsevier ENTCS, 1998.
14. A. D. Gordon, P. D. Hankin, and S. B. Lassen. Compilation and equivalence of imperative objects. In *Proc. of FST & TCS '97*, volume 1346 of *LNCS*, Dec. 1997.
15. R. Harper and M. Lillibridge. Polymorphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6:361–380, 1993.
16. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. of ECOOP '91*, volume 512 of *LNCS*, pages 133–147, 1991.
17. H. Hüttel and J. Kleist. Objects as mobile processes. TR RS-96-38, BRICS, Oct. 1996.
18. J. Kleist and D. Sangiorgi. Imperative objects and mobile processes. In *Proc. of PROCOMET '98*. North-Holland, 1998.
19. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proc. of ICALP '98*, volume 1443 of *LNCS*, 1998.
20. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*, volume 623 of *LNCS*, pages 685–695, 1992.
21. U. Nestmann, H. Hüttel, J. Kleist, M. Merro. Aliasing Models for Object Migration. In *Proc. of Euro-Par '99*, volume 1685 of *LNCS*, pages 1353–1368, 1999.
22. B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In *Proc. of TPPP '94*, volume 907 of *LNCS*, pages 187–215, 1995.
23. D. Sangiorgi. An interpretation of typed objects into typed π-calculus. TR 3000, INRIA, 1996.
24. V. T. Vasconcelos. Typed concurrent objects. In *Proc. of ECOOP '94*, volume 821 of *LNCS*, pages 100–117, 1994.
25. R. Viswanathan. Full abstraction for first-order objects with recursive types and subtyping. In *Proc. of LICS '98*, pages 380–391, 1998.

A Higher-Order Specification of the π-Calculus

Joëlle Despeyroux

INRIA,
2004 Route des Lucioles, B.P. 93, F-06902 Sophia-Antipolis Cedex, France.
Joelle.Despeyroux@inria.fr

Abstract. We present a formalization of a typed π-calculus in the Calculus of Inductive Constructions. We give the rules for type-checking and for evaluation and formalize a proof of type preservation in the Coq system. The encoding of the π-calculus in Coq uses Coq functions to represent bindings of variables. This kind of encoding is called a higher-order specification. It provides a concise description of the calculus, leading to simple proofs. The specification we propose for the pi-calculus formalizes communication by means of function application.

1 Introduction

The π-calculus [MPW92,Mil91] is a model of concurrent computation based upon the notion of naming. Processes receive and emit names, which denote channels.

We propose here new formalizations of both evaluation and typing rules for a typed π-calculus, with the aim to enable machine-checked proofs of various properties of languages based on this calculus. As a simple, typical illustration of such a proof, we give a proof of preservation of types, also called the subject-reduction property, for the calculus.

For our experiment, we chose the Coq system [BBC+97,HKPM97], based on the Calculus of Inductive Constructions [CH88,PM92]. In this system, proofs are performed by applying tactics, in a goal-directed manner. We like this system because of its rich meta-language (a higher-order functional language, allowing full dependent inductive types), and because it is based on Type Theory.

Thanks to the use of the *higher-order abstract syntax* technique, giving rise to so-called *higher-order specifications*, our formalization of the π-calculus does not require definitions for free or bound variables in a term. Nor does it require definitions of notions of substitutions, which are implemented using the meta-level application, i.e. application available in the Logical Framework used to implement our calculus (which in our case is the Calculus of Inductive Constructions).

Robin Milner [Mil91] introduced notions of *abstractions* and *concretions* leading to a presentation of the calculus in which input processes evaluate to functions and output processes evaluate to concretions, i.e. (intuitively) pairs of a value and a process. Our formalization starts from this idea, replacing concretions by higher-order functions, thus leading to a nicer, more uniform presentation, in which communication is formalized by function application.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 425–439, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Related works include formalizations of various π-calculi in the same system. These works were conducted with different goals in mind. Some of them had the same goals as ours. The others aimed at the study of bisimulation techniques. We shall discuss both kinds of related works in detail at the end of the paper.

Note on the Coq syntax. Terms in Coq are terms of a typed λ-calculus, where the abstraction $\lambda x : t.a$ is written [x:t]a, the dependent product type $\forall x : t.a$ is written (x:t)a, while the non-dependent product type $A \to B$ is simply written $A \to B$. The application of a function f to x is written $(f\ x)$. The expression $\exists x : t.a$ is written (Ex [x:t]a). The type of sets is denoted as Set while the type of propositions in denoted as Prop. As usual in Type Theory, the type $A_1 \to \cdots \to A_n \to B$ where B is a proposition represents (through the Curry-Howard isomorphism) the inference rule with premisses $A_1, \cdots, A_n$ and conclusion B. Inductive types are defined by a list of constructors together with their types, as for example for the naturals:

```
Inductive nat : Set := 0 : nat | S : nat -> nat.
```

An inductive definition automatically generates both an inductive principle for the type being defined and elimination principles which enable the definition of recursive functions on this type. In the following, Coq text will be given as above in a typewriter font.

2 Syntax

The terms of the π-calculus are processes, which receive and emit names denoting channels. For the sake of simplicity, we do not consider here the operators of choice and matching. It will be straightforward to add them.

2.1 The Monadic π-Calculus

Processes are defined by the following rule:

$$\text{Processes}: \quad P, Q ::= \bar{x}y.P \mid x(y).P \mid \nu x : t.P \mid (P \mid Q) \mid !\ P \mid 0$$

in which x and y belong to an enumerable set of names and t denotes the type of a name as described in section 5.

The expression $\bar{x}y.P$ denotes a process which sends a name y along the channel x (*output* operation), then behaves as P. The term $x(y).P$ denotes a process which receives a name z along the channel x (*input* operation), then behaving like $P[z/y]$ (P in which z replaces y). The expression $(P \mid Q)$ denotes the *parallel* composition of P and Q where P and Q are concurrently active. 0 denotes the *empty* process. $!\ P$ denotes the infinite process $P \mid P \mid \cdots$; the ! operator (pronounce "bang") is called the *replication* operator. Finally, $\nu x : t.P$ declares x as a private channel with type t in P. The ν operator is called a *restriction* as it restricts the use of the name x to P.

In the term $\nu x : t.P$, the name x is bound in the term P. Similarly, in the term $x(y).P$, the name y is bound in the term P, waiting to be substituted by another name before P can be executed.

Processes evaluate to other processes while performing *actions*. In the standard approach, there are four kinds of actions: input actions xy (input a value for variable y on a channel x), output actions $\bar{x}y$ (output value y on a channel x), bounded output action $\bar{x}(y : t)$, and silent actions τ. Communications are always silent.

$$\text{Actions}: \quad A ::= xy \mid \bar{x}y \mid \bar{x}(y : t) \mid \tau$$

An original, and central notion of the π-calculus is the notion of *scope extrusion*. A name is said to be extruded when it is a private name sent to a process. In such a case, the scope of the name is extended to the new process which receives it. This phenomenon, which may make the notion of binding in the π-calculus unclear at first glance, will be illustrated and discussed later on.

The informal account of terms requires the definition of the set of variables in a term, that we denote as $Var(t)$. It also requires the definition of free variables ($F(t)$) and bound variables ($B(t)$) in a term. In the informal presentation of the calculus, we will need a notion of substitution of a variable for another variable in a term, written as usual as $P\{y/x\}$.

2.2 Formalization of the Syntax

We encode the terms of the calculus in Coq as follows. First we introduce a new parameter *name*, and suppose two usual properties on the set of names, that we state as axioms:

$$\forall x, y : name.\ x = y \vee x \neq y; \quad \forall x : name. \exists y : name.\ x \neq y$$

```
Parameter name: Set.
Axiom name_decidable : (x,y:name) (x=y) \/ (~x=y).
Axiom name_different : (x:name)(Ex [y:name](~x=y)).
```

Then we introduce the set of types as a parameter too, stating some properties on this set later on.

```
Parameter typ : Set.
```

The processes are then defined as follows:

```
Inductive proc : Set :=
    Nil  : proc
  | In   : name -> (name -> proc) -> proc
  | Out  : name -> name -> proc -> proc
  | Par  : proc -> proc -> proc
  | Bang : proc -> proc
  | Res  : typ  -> (name -> proc) -> proc.
```

The representation of processes requires some explanations. There are two binding operators in the π-calculus: the input and the restriction operator. We use here Coq functions to represent bindings of variables; hence the type $(name \rightarrow proc)$ in the types of input processes and restrictions. This kind of encoding is called a *higher-order specification*. This method of formalization, called the higher-order abstract syntax technique, provides an elegant way to formalize the binding nature of the constructors. In particular, it provides for free a definition of terms up to α-*equivalence*. This method also provides an elegant way to formalize *substitutions*, as we shall see later on, when formalizing the rules for evaluation.

Note that the type name is declared as a parameter, not as a variable. This prevents name to be instantiated by an inductive set.

A restricted name never gets instantiated. Nevertheless, using functions (i.e. binding operations) to represent the restriction yields a simple and elegant formalization of the notion of private names.

In our description of the π-calculus, the only visible argument of an action will be the name of a channel. In other words, actions only need to carry a name. As a consequence, there is no need for bound output actions in our development:

```
Inductive action : Set :=
    InA : name -> action | OutA : name -> action | Tau : action.
```

3 Simple Examples of Evaluation

We give here some simple examples of communication. The simplest case of communication is the following one, where a name y is sent to Q, along a channel x:

$$\bar{x}y.P \mid x(z).Q \xrightarrow{\tau} P \mid Q\{y/z\}$$

Then let us illustrate the phenomenon of scope extrusion. This occurs in a communication where a process sends a private name to an external process, as in:

$$\nu y : t.(\bar{x}y.P) \mid x(z).Q \quad \text{which reduces to} \quad \nu y : t.(P \mid Q[y/z]).$$

The private name y has been passed to the external process Q. We say that P has extruded the scope of the private channel y. We have a similar phenomenon in imperative languages when a local name y is sent to a procedure Q taking its parameters by reference.

4 Evaluation Rules

The evaluation rules define a judgement $P \xrightarrow{a} Q$ meaning that P evaluates to Q producing an action a. The chosen semantics is the early transition semantics.

4.1 Informal Evaluation Rules

We need here the definition of all variables in a term ($Var(t)$), together with the definitions of fresh and bound variables ($F(t)$ and $B(t)$). Moreover, usual informal presentations of the calculus use the open and close rules, which *open and close the scope of a name*, in order to describe the phenomenon of scope extrusion that we illustrated above (section 3). The informal description of the evaluation rules is the following one.

input :
$$x(y).P \xrightarrow{xz} P\{z/y\}$$

output :
$$\bar{x}y.P \xrightarrow{\bar{x}y} P$$

com :
$$\frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}y} Q'}{Q \mid P \xrightarrow{\tau} Q' \mid P'}$$

open :
$$\frac{P \xrightarrow{\bar{x}y} P'}{\nu y : t.P \xrightarrow{\bar{x}(y:t)} P'} \quad x \neq y$$

close :
$$\frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{P \mid Q \xrightarrow{\tau} \nu y : t.(P' \mid Q')} \quad y \notin F(P) \qquad \frac{P \xrightarrow{xy} P' \qquad Q \xrightarrow{\bar{x}(y:t)} Q'}{Q \mid P \xrightarrow{\tau} \nu y : t.(Q' \mid P')} \quad y \notin F(P)$$

res :
$$\frac{P \xrightarrow{a} P'}{\nu x : t.P \xrightarrow{a} \nu x : t.P'} \quad x \notin Var(a)$$

par :
$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \quad B(a) \cap F(Q) = \emptyset \qquad \frac{P \xrightarrow{a} P'}{Q \mid P \xrightarrow{a} Q \mid P'} \quad B(a) \cap F(Q) = \emptyset$$

bang :
$$\frac{!P \mid P \xrightarrow{a} P'}{!\, P \xrightarrow{a} P'}$$

In the above set of rules, rules for *input* and *output* describe the basic cases of input and output processes. The *com* and *close* rules describe communication, while the rest of the rules describe the propagation of evaluation through a restriction, a parallel composition, or a replication.

The replication rule could be simpler. The advantage of the chosen rule is that it enables the extension of the rules with a choice operator.

4.2 Formalization of the Evaluation Rules

We shall take full advantage here from our choice to use functions in the formalization of the syntax.

First, as input processes are described as functions, we just say that input processes evaluate to themselves. This follows Robin Milner approach [Mil91] where he introduced a notion of *abstraction* to denote the result of the evaluation

of input processes. We will define an inductive predicate *evali* for processes performing an input action:

```
Inductive evali : proc -> action -> (name -> proc) -> Prop
```

with as first and simplest case the case for input processes:

```
evali_in : (x:name)(p:name->proc) (evali (In x p) (InA x) p)
```

Robin Milner also introduced a notion of *concretion* [Mil91], which (intuitively) was a pair of a value and a process, to denote the result of the evaluation of output processes. Then he (informally) described the communication as the "application" of an abstraction to a concretion. We thought it would be nice, in order to formalize the communication by a real application, to describe the evaluation of an output process as yielding a *higher-order function* which will then be applied to the result of the evaluation of an input function in order to describe a communication, as we shall see later on.

Let us say that the evaluation of output processes yields higher-order functions taking functions of type $(name \rightarrow proc)$ as arguments. We define an inductive predicate *evalo* for processes performing an output action:

```
Inductive evalo : proc -> action -> ((name->proc) -> proc) -> Prop
```

with as first and simplest case the case for output processes:

```
evalo_out : (x,y:name)(p:proc)
   (evalo (Out x y p) (OutA x) [f:name->proc](Par (f y) p))
```

Example We are now in a position to see how we can describe communication on a basic example:

$$\frac{P \stackrel{\bar{x}y}{\rightarrow} P' \qquad Q \stackrel{xy}{\rightarrow} Q'}{\bar{a}b.P \mid a(y).Q \stackrel{\tau}{\rightarrow} P' \mid Q'\{b/y\}}$$

The evaluation of the input process $a(y).Q$ gives the function Q itself. Then the result of the evaluation of the output $\bar{a}b.P$ is a function $\lambda f : name \rightarrow proc.((f\ b) \mid P)$. Applying this function to Q yields the expected result: $(Q\ b) \mid P$.

```
(Out a b P)  --(OutA a)--> [f:name->proc]((f b) | P) = P'
   (In a Q)   --(In a)-->  Q
-------------------------------------------------------
   (Out a b P) | (In a Q) --Tau--> (P' Q) = (Q b) | P
```

Thus the evaluation of both input and output processes yields functions. Communication is formalized by applying the result of the evaluation of the output process to the result of the evaluation of the input process. Our formalization has replaced concretions by higher-order functions, thus leading to

a nicer, more uniform presentation, in which communication is formalized by a real application of a function. On a technical level, the higher-order abstract syntax method enables us to describe *substitution* in evaluation rules by simply using the *application* of a Coq function to terms -names in this case.

It is staightforward to prove (on paper) the consistency of this presentation of the semantics of the π-calculus with functions with respect to the standard approach.

The rule for example for left communication will be formalized as follows, in a third set of inductive rules describing the evaluation of processes that perform silent actions:

```
Inductive eval :  proc -> action -> proc -> Prop :=
  | eval_coml:
      (p:proc)(x:name)(p':(name->proc)->proc)
      (evalo p (OutA x) p') ->
      (q:proc)(q':name->proc) (evali q (InA x) q') ->
      (eval (Par p q) Tau (p' q'))
  [...].
```

Note that communication is always simply formalized by an application, no matter whether names are extruded or not during the communication. This is the reason why we do not need rules for opening and closing the scope of names in our formalization.

The other rules -for the parallel, replication and restriction operators- need to be duplicated, one for each type of action involved (input, output or silent). Note however that all proofs, either formal or informal, proceed by a case analysis on the type of the action involved anyway. Thus this apparent defect should not be considered as a drawback of our formalization.

Thanks to the use of the higher-order abstract syntax technique, when evaluating a process which declares a private name, we do not need to check that this private name is a fresh name. The freshness of the name naturally comes from the use of a universal quantification on names in the premiss of the rule.

```
eval_res :
      (u:typ)(p:name->proc)(a:action)(q:name->proc)
      ((x:name) (eval (p x) a (q x))) ->
      (eval (Res u p) a (Res u q)).
```

The rules for parallel composition do not require any side condition regarding bound variables either. This is because the actions do not carry values here, but only the name of a channel.

```
eval_parl : (p:proc)(a:action)(p':proc)
      (eval p a p') -> (q:proc) (eval (Par p q) a (Par p' q))
```

The rule for replication (omitted here) is straightforward.

5 Type Checking Rules

The type system that we chosed uses a notion of *directionality* of names [PS95], which is their ability to be used as emitters, receivers or both (or none). This name usage discipline is implemented by type-checking rules. This type system, while being simple, is at the same time widely used as witnessed by the experiment of Pict [PT97] and non-trivial because of the sub-typing that it captures. For our purpose, it is more interesting than the type system for the polyadic π-calculus [Mil91], whose typing rules are totally straightforward.

5.1 Types

A type t of a name of a channel is the union of its *capacity*, which represents how the channel can be used (read, write, both or none) and the *types* of the names that can transit in it:

$$\begin{array}{ll} \text{Capacities}: & c ::= \mathit{read} \mid \mathit{write} \mid \mathit{both} \mid \mathit{none} \\ \text{Types}: & t ::= (c, t) \end{array}$$

We formalize the capacities of names as follows, where none is called top (for top of the mini-lattice):

```
Inductive cap : Set :=
  read : cap | write : cap | both : cap | top : cap.
```

Remember we only defined types as:

```
Parameter typ : Set.
```

This means that we do not impose a particular implementation of types. Instead we state some properties this implementation must enjoy. First we must be able to extract both the capacity $\langle t \rangle$ and the type $[t]$ parts of a type t:

```
Parameter get_cap : typ -> cap.
Parameter get_typ : typ -> typ.
```

Then we have a subtyping relation on capacities, denoted as $\leq$ and defined as follows:

$$c \leq c \mid \mathit{both} \leq c \mid c \leq \mathit{none}$$

The subtyping relation on capacities induces a subtyping relation on types, that we still denote as $\leq$ and define as follows:

$$\begin{array}{c} t \leq t \\ \langle t' \rangle = \mathit{none} \Rightarrow t \leq t' \\ \langle t \rangle \leq \mathit{read} \ \wedge \langle t' \rangle = \mathit{read} \ \wedge [t] \leq [t'] \Rightarrow t \leq t' \\ \langle t \rangle \leq \mathit{write} \wedge \langle t' \rangle = \mathit{write} \wedge [t'] \leq [t] \Rightarrow t \leq t' \end{array}$$

The $\leq$ predicates are formalized by the following inductive types, the obvious rules of which we omit here:

```
Inductive stype_cap : cap -> cap -> Prop
Inductive stype : typ -> typ -> Prop
```

The covariance of the read capability and the contravariance of the write capability introduce a subtyping in depth, as the typing rules will show.

A process is said to be well *typed in a particular environment* which gives types to each of its free name. In contrary to the usual case in typed programming languages, the type-checking rules do not give types to terms in the π-calculus.

5.2 Examples

The process $\bar{x}y \mid x(u).\bar{u}v \mid \bar{x}z$ is well typed in an environment where x has the capabilities *both* (both read and write). $\bar{x}y$ alone is well typed in an environment where x has a capabilities $c \leq write$, which means that c is either *both* or *write*.

The following example shows how certain interferences between processes can be eliminated. Consider the process $P = \nu y : (both, t).(\bar{x}y.y(z))$, where x has type $(write, (write, t))$. The process P gives the writing capacity to the outside, while keeping the reading capacity for himself on the name y. After the transmission of y, a process sending a value to P on channel x is sure that this value cannot be captured by another process.

The previous example also illustrates the use of subtyping in depth.

5.3 Informal Typing Rules

The informal typing rules, defining a judgement $env \vdash proc$, are self-explanatory. (We omit the obvious rules for par and bang.) Note the need for the Var predicate in the input and res rules.

$$typ_nil : \quad \Gamma \vdash 0$$

$$typ_input : \quad \frac{\langle \Gamma(x) \rangle \leq read \qquad \forall z \notin Var(P)\ \Gamma.(z : [\Gamma(x)]) \vdash P\{z/y\}}{\Gamma \vdash x(y).P}$$

$$typ_output : \quad \frac{\langle \Gamma(x) \rangle \leq write \qquad \Gamma(y) \leq [\Gamma(x)] \qquad \Gamma \vdash P}{\Gamma \vdash \bar{x}y.P}$$

$$typ_res : \quad \frac{\forall y \notin Var(P)\ \Gamma.(y : t) \vdash P\{y/x\}}{\Gamma \vdash \nu x : t.P}$$

5.4 Formalization of the Typing Rules

An environment is encoded as a function from names to types:

```
Definition env : Type := name -> typ.
```

Then the type-checking rules are a straightforward translation from informal rules, where the only interesting point to note is that we do not need to look for fresh names in the rules for input and restriction. The freshness condition is implemented by an universal quantification on variables in the premisses involved.

The complete set of rules is as follows:

```
Inductive type : env -> proc -> Prop :=
    type_nil : (g:env) (type g Nil)
  | type_in  :
       (g:env)(x:name)(p:name->proc)
       (stype_cap (get_cap (g x)) read) ->
       ((y:name) ((g y)=(get_typ (g x))) -> (type g (p y))) ->
       (type g (In x p))
  | type_out :
       (g:env)(x,y:name)(p:proc)
       (stype_cap (get_cap (g x)) write) ->
       (stype (g y) (get_typ (g x))) ->
       (type g p) ->
       (type g (Out x y p))
  | type_res :
       (g:env)(t:typ)(p:name->proc)
       ((y:name) ((g y)=t) -> (type g (p y))) ->
       (type g (Res t p)).
```

6 Preservation of Types

We have to prove three theorems, one for each kind of actions. The theorem for the silent actions simply states the following:

```
Theorem sr_tau:
   (p,q:proc) (eval p Tau q) -> (g:env) (type g p) -> (type g q).
```

The proof of this theorem uses the following natural property for communications. The property says that a communication between two processes resulting from the evaluation of well-typed processes is well-typed:

```
Theorem sr_com:
  (p:proc)(x:name)(p':(name->proc)->proc)
  (evalo p (OutA x) p') -> (g:env) (type g p) ->
  (q:proc)(q':name->proc) (evali q (InA x) q') -> (type g q) ->
  (type g (p' q')).
```

The proof of the above theorem needs the theorems stating the preservation of types properties for input and output actions:

```
Theorem sr_in:
   (p:proc)(a:action)(q:name->proc)
   (evali p a q) -> (x:name)(a=(InA x)) ->
   (g:env) (type g p) ->
   (y:name) (stype (g y) (get_typ (g x))) -> (type g (q y)).

Theorem sr_out:
   (p:proc)(a:action)(q:(name->proc)->proc) (evalo p a q) ->
   (x:name)(a=(OutA x)) -> (g:env) (type g p) -> (f:name->proc)
   ((y:name) (stype (g y) (get_typ (g x))) -> (type g (f y))) ->
   (type g (q f)).
```

Note in the above theorems how we deal with the typing of functions. We say that a function f from name to processes is well-typed if the process $(f\ y)$ is well-typed for every name y of the appropriate type. Similarly, a function F from (name to processes) to processes will be said to be well-typed if the process $(F\ f)$ is well-typed for every well-typed function f.

Except for some simple lemmas on the subtyping relation, no further lemmas are required in the proof. In particular there is no need for tedious proofs of lemmas about renaming, freshness of variables and the like, contrary to almost all the other developments we have seen so far.

7 Related Work

Related works include several recent formalizations of various π-calculi, all done in the Coq system. As we mentioned in the introduction of the paper, some of these works had the same goals as ours, while others aimed at the study of bisimulation techniques. To be fair with the experiments in the latter case, we must say that the use of functions in our specification of the π-calculus, while providing excellent results for our goals, might make the proofs of correctness of bisimulation techniques a bit harder than usual.

An extensive implementation of the π-calculus, including proofs of correctness of bisimulation techniques, has been realized by Daniel Hirschkoff [Hir97,Hir99] using the de Bruijn method. Our contribution is much more modest in the sense that it only provides a new basis for such a study. However, the de Bruijn technique quickly leads to very obscure semantic descriptions and makes proofs long and tedious. 75 percent of the development by Daniel Hirschkoff concern manipulation of de Bruijn codes, which is not the subject of interest.

A different method has recently been investigated by Guillaume Gillard [Gil00], on a concurrent object calculus recently proposed by Andrew Gordon and Paul Hankin. Guillaume Gillard provides a proof of preservation of types for this calculus. The method used is due to Andrew Gordon [Gor94]. It consists

in describing terms modulo alpha-conversion, over a predefined set of λ-terms implemented with de Bruijn codes. The part of the development devoted to the manipulation of de Bruijn terms, while still sizeable, is definitely more reasonable than in the standard approach.

A recent development has been realized by Loic Henry-Greard [Hen98]. The proof done is the same as ours: the proof of preservation of types for the monadic π-calculus. The proof uses a (slight extension of a) method due to Randy Pollack and James Mc Kinna, extensively used to formalize Lego in Lego. The basic idea of the method is to distinguish between bound variables (represented by variables) and free variables (represented by parameters). This gives a nice solution to the problem of capture-avoiding substitution. However, there is still an overhead in both the descriptions and the proofs, to manipulate the variables and the parameters. It is interesting to note that our proof, while being considerably shorter, has exactly the same structure as Loic Henry-Greard's proof.

Finally, Furio Honsell, Marino Miculan and Ivan Scagnetto have proposed a higher-order description of the π-calculus (with the match operator) [HMS99], different from ours, which they have used in the proofs of correctness of various bisimilarity laws. Their formalisation uses meta-level (Coq) functions to encode the input and restriction operators like us; However, the rest of the presentation greatly differs from ours. They need to duplicate the evaluation rules in a non-natural way: some of the rules yield processes while the others yield functions from name to processes. Communication is formalized using open and close rules, which, despite the use of higher-order syntax, require a freshness predicate. Meanwhile, they provide a formal study of some bisimilarity principles, which, as it uses rules of evaluation (partly) described by means of functions, should a priori inspire a similar study using our specification.

8 Conclusion and Future Work

We have presented in this paper a higher-order formalization of a monadic π-calculus in the Calculus of Inductive Constructions, and provided a proof of type preservation for this calculus in the Coq system. The specification we proposed formalizes communication by a function application. Our proof is very short: it is only three pages long. We believe that the same proof performed on alternative descriptions of the same calculus would give much longer and more tedious proofs. Extending our development to the polyadic π-calculus, while requiring longer proofs, should be straightforward.

It might be worth noting that the technique of higher-order abstract syntax has been rarely used to describe imperative languages up to now. It seems that this method, while being of great benefit in the descriptions of functional programming languages, is a bit less suitable for imperative languages. However, the π-calculus is considered as an imperative language, as communication is described as a side-effect in the meta-theoretical studies of the calculus. We may hope that the specification proposed here might give new ideas for the description of imperative languages.

Another interesting point to note is that only a limited form of higher-order abstract syntax is required to describe first-order, usual π-calculi. The point is that we only need functions from name to processes to describe binders, while functions from processes to processes would be required in the description of the higher-order π-calculus, where processes themselves can be passed on a channel. The problem is that such types are not allowed as types of constructors of an inductive type for processes, at least in the standard way as the one provided by almost all the current systems providing induction. Consequently, we are not able to use the technique of higher-order abstract syntax to describe the higher-order π-calculus today. However, there are already propositions for various meta-logics in the literature [DPS97,DL98,MM97,Hof99] which could be used as a basis for the implementation of the system we need.

Concerning future work, our main goal is to provide the basis for a machine-checked study of languages based on the π-calculus. Particularly relevant here are proofs of properties such as the preservation of types for languages for which the rich type system make those proofs challenging, even in their hand-written presentations. We think here of languages involving polymorphism as the Pict [PT97] or the Join [FGL+96] languages, or to type-checking rules to prevent dead-lock as proposed in [Kob98]. Other interesting languages will be those arising from the various current propositions for concurrent and object calculi [Bou97,GH98].

As we already mentioned, future work of a different nature is the study of the theory of the π-calculus: bisimilarity techniques. It would be very interesting to see whether the higher-order description of the π-calculus we propose here can be of real benefit too in such formal studies.

In the long term, we hope such formal studies to be used to formalize proof of correctness of programs written in languages based on the π-calculus.

Acknowledgements. Thanks go to Davide Sangiorgi who introduced us to the π-calculus. Our description of the calculus comes out of many discussions we had, trying to understand and formalize the binding mechanism in the π-calculus. We also thank the anonymous reviewers for their constructive comments.

References

[BBC+97] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, C. Muñoz, C. Murthy, C. Parent, C. Paulin, A. Saïbi, and B. Werner. The Coq Proof Assistant Reference Manual – Version V6.1. Technical Report 0203, INRIA, August 1997.

[Bou97] Gérard Boudol. The pi-calculus in direct style. In *proceedings of the POPL ACM Conference on Principles of Programming Languages*, pages 228–241, 1997.

[CH88] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[DL98] Joëlle Despeyroux and Pierre Leleu. A modal λ-calcul with iteration and case constructs. In *proceedings of the annual Types for Proofs and Programs seminar*, Springer-Verlag LNCS 1657, March 1998. An extended and revised version will appear in the forthcoming special issue of MSCS on Intuitionistic Modal Logics and Applications.

[DPS97] Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. Primitive recursion for higher-order abstract syntax. In Philippe de Groote and J. Roger Hindley, editors, *proceedings of the TLCA 97 Int. Conference on Typed Lambda Calculi and Applications, Nancy, France, April 2–4*, pages 147–163. Springer-Verlag LNCS 1210, April 1997. An extended version is available as CMU Technical Report CMU-CS-96-172. An extended and revised version will appear in TCS (Theoretical Computer Science).

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proc. of the CONCUR conference*, 1996.

[GH98] A. Gordon and P. Hankin. A concurrent object calculus: reduction and typing. In *proceedings of the HLCL Conference*, 1998.

[Gil00] Guillaume Gillard. A formalization of a concurrent object calculus up to alpha-conversion. In *Proceedings of the 17th International Conference on Automated Deduction, CADE-17*, June 2000.

[Gor94] A. Gordon. A mechanisation of name-carrying syntax up to alpha-conversion. In *proceedings of the int. workshop on Higher Order Logic Theorem Proving and its Applications, Vancouver*, Springer-Verlag LNCS 780, pages 414–426, 1994.

[Hen98] Loïc Henry_Greard. A proof of type preservation for the pi-calculus in Coq. Research Report RR-3698, Inria, December 1998. Also available in the Coq Contrib library.

[Hir97] Daniel Hirschkoff. A full formalization of pi-calculus theory in the Calculus of Constructions. In Elsa Gunter, editor, *proceedings of the International Conference on Theorem Proving in Higher Order Logics*, Murray Hill, New Jersey, August 1997.

[Hir99] Daniel Hirschkoff. *Mise en oeuvre de preuves de bisimulation*. Phd thesis, École Nationale des Ponts et Chaussées (ENPC), January 1999. In French.

[HKPM97] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The coq proof assistant: a tutorial, version 6.1. Technical Report RT 0204, Inria, Rocquencourt, France, August 1997. page html: http://www.inria.fr/RRRT/RT-0204.html.

[HMS99] Furio Honsell, Marino Miculan, and Ivan Scagnetto. Pi-calculus in (co)Inductive Type Theory. *to appear in TCS*, 1999.

[Hof99] Martin Hofmann. Semantical analysis of higher-order abstract syntax. In IEEE, editor, *proceedings of the International Conference on Logic In Computer Sciences, LICS*, 1999.

[Kob98] Naoki Kobayashi. A partially deadlock-free typed process calculus. *ACM Transactions on Programming Languages and Systems*, 20-2, December 1998. A preliminary summary appeared in LICS'97.

[Mil91] Robin Milner. The polyadic pi-calculus: a tutorial. Technical Report ECS-LFCS-91-180, LFCS, Edinburgh University, October 1991.

[MM97] Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax: An extended abstract. In Glynn Winskel, editor, *Proceedings of the Twelfth Annual Symposium on Logic in Computer Science*, Warsaw, Poland, June 1997. To appear.

[MPW92] R. Milner, R. Parrow, and J. Walker. A calculus of mobile processes, (part I and II). *Information and Computation*, 100:1–77, 1992.

[PM92] Christine Paulin-Mohring. Inductive definitions in the system coq. rules and properties. In J.F. Groote M. Bezem, editor, *Proceedings of the International Conference on Typed Lambda Calculi and Applications, TLCA'93*, Springer-Verlag LNCS 664, pages 328–345, 1992. also available as a Research Report RR-92-49, Dec. 1992, ENS Lyon, France.

[PS95] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 11, 1995.

[PT97] Benjamin Pierce and David Turner. Pict: A programming language based on the pi-calculus. Technical Report ., IU, 1997.

Open Ended Systems, Dynamic Bisimulation and Tile Logic

Roberto Bruni[1], Ugo Montanari[1], and Vladimiro Sassone[2]

[1] Dipartimento di Informatica, Università di Pisa, Italia.
[2] Dipartimento di Matematica e Informatica, Università di Catania, Italia.
{bruni,ugo}@di.unipi.it, vs@dmi.unict.it

Abstract The SOS formats ensuring that bisimilarity is a congruence often fail in the presence of structural axioms on the algebra of states. Dynamic bisimulation, introduced to characterize the coarsest congruence for CCS which is also a (weak) bisimulation, reconciles the bisimilarity as congruence property with such axioms and with the specification of open ended systems, where states can be reconfigured at run-time, at the cost of an infinitary operation at the meta-level. We show that the compositional framework offered by tile logic is suitable to deal with structural axioms and open ended systems specifications, allowing for a finitary presentation of context closure.

Keywords: Bisimulation, SOS formats, dynamic bisimulation, tile logic.

Introduction

The semantics of dynamic systems can be conveniently expressed via labelled transition systems (LTS) whose states are terms over a certain algebra and whose labels describe some abstract behavioral information. Provided such information models the possible interactions between various components, the framework yields a compositional semantics. Plotkin's *structured operational semantics* (SOS) [23] is one of the most successful such frameworks, where the transitions a system can perform are defined by recursion on the structure of its states.

Several notions of equivalence on the state space of LTSs have been considered in the literature that take into account particular aspects. For example, if one is interested only in the action sequences performed by the system, then one should observe *traces*, whereas in a truly concurrent approach one would rather observe *partial orderings* of actions. State equivalences can then be defined on the basis of the chosen observables. In this paper we consider *bisimulation* equivalences [18,22] (with *bisimilarity* meaning the maximal bisimulation), where the entire branching structure of the transition system is accounted for: two states are equivalent if whatever transition one can perform, the other can simulate it via a transition with the same observation, still ending in equivalent states.

Research supported by CNR Integrated Project *Progettazione e Verifica di Sistemi Eterogenei Connessi mediante Reti*; by Esprit Working Groups *CONFER2* and *COORDINA*; by TMR Network *GETGRATS*; and by MURST project *TOSCA*.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 440–456, 2000.
© Springer-Verlag Berlin Heidelberg 2000

One of the main advantages of a compositional semantics is that each subcomponent of a state can be safely replaced by any equivalent subcomponent without affecting the overall behavior. This triggered many efforts devoted to the study of SOS *formats* whose syntactic constraints guarantee that *bisimilarity is a congruence*. Among the most popular such formats, we mention the simple *De Simone* format [10], the more general *positive* GSOS format [3], and the 'liberal' family of `tyft` formats [13,1] (`tyxt`, `zyft`, *promoted* `tyft`,...). Yet, in many interesting cases the use of these formats is not straightforward. For instance, although it is often convenient to have some structural axioms on states, these cannot be handled by ordinary formats. Fall in this case several semantic descriptions based on the *chemical abstract machine* [2], where operators are often assumed to be associative, commutative and with unit, as e.g., the parallel composition of CCS. Further examples are given by systems modeled using UML graphs, that must be taken up to suitable isomorphisms and therefore require the LTS to be defined on suitable structural equivalence classes. Also the (finite) π-calculus [19] is defined by rules in good format when substitution is explicit, but agents are subject to substitution axioms.

It is therefore often necessary to resort to the largest congruence included in the bisimilarity by an operation of closure 'for all contexts', which results in a congruence which is no longer a bisimulation. *Dynamic bisimulation* [20], instead, performs such a context closure during the bisimulation 'game'. Dynamic bisimilarity was shown to capture the coarsest equivalence for CCS agents among weak bisimulations that are also congruences, being completely axiomatized by the axioms of strong observational equivalence plus two of the three Milner's τ-laws. The basic idea is to allow at every step of bisimulation not only the execution of an action, but also the embedding of the two states under comparison within the same, but otherwise arbitrary, context. It is worth remarking that such dynamical contextual embedding has a natural interpretation in terms of dynamic reconfiguration of the system, and hence can find many application in practice for modeling *open ended systems*. Its main drawback is the lack of a convenient representation at the level of the SOS rules; it rather has the spirit of a 'meta' construction, involving a *universal* quantification on contexts. An interesting approach – in the style of Sewell's work on defining LTS from reduction systems [24] – would be to enrich the transition system with all (unary) contexts as labels, and all transitions $p \xrightarrow{C[_]} C[p]$ for any state p and context $C[_]$. This solution, however, would still be at the meta-level and infinitary in principle.

In this paper we propose to recast dynamic bisimulation inside the *tile model*, where it can be finitely modeled. The tile model [12] is a formalism for modular descriptions of the dynamic evolution of concurrent systems. It relies on a form of rewrite rules with side effects, called *basic tiles*, which are reminiscent of both SOS rules and *context systems* [14], collecting intuitions coming from *structured transition systems* [9] and *rewriting logic* [16]. In particular, by analogy with rewriting logic, the tile model can be defined employing a logical presentation, called *tile logic*, where tiles are (decorated) sequents subject to inference rules.

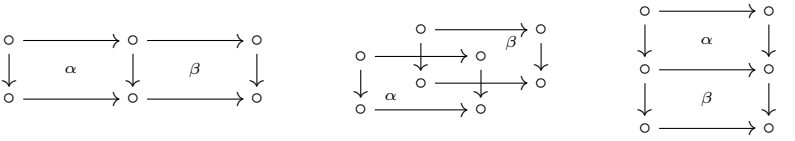
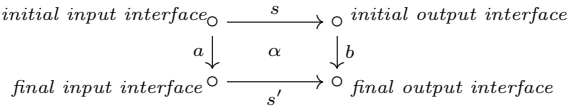


Figure 1. Horizontal, parallel and vertical tile compositions.

Tile logic. Tile logic extends rewriting logic (in the unconditional case) by taking into account state changes with side effects and rewriting synchronization. Basically, a set of rules describes the behaviour of certain (partially specified, in the sense that can contain variables) *configurations*, which may interact through their *interfaces*. Then, the behaviour of a system as a whole consists of a coordinated evolution of its sub-systems. The name 'tile' is due to the graphical representation of such rules, which have the form



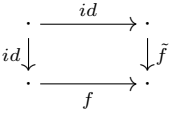
also written $\alpha : s \xrightarrow{a}_{b} s'$, stating that the *initial configuration* s of the system evolves to the *final configuration* s' via the tile α, producing the *effect* b, which can be observed by the rest of the system. However, such a step is allowed only if the subcomponents of s (i.e., the arguments of the *context* s) evolve to the subcomponents of s', producing the effect a, which acts as the *trigger* of α. Triggers and effects are called *observations* and tile vertices are called *interfaces*. The arrows s, a, b and s' give the *border* of α.

Tiles are a natural model for reactive systems that are *compositional in space* – the behaviour of structured states can be defined as a coordination of the activities of the subcomponents – and *compositional in time*, as they offer a powerful framework for modelling the composition of computations. Indeed, tiles can be composed horizontally, vertically, and in parallel to generate larger steps. The operation of parallel composition corresponds to building concurrent steps, where two (or more) disjoint configurations can concurrently evolve. Of course, the border of a concurrent step is the parallel composition of the borders of each component of the step. Horizontal composition yields rewriting synchronization: the effect of the first tile provides the trigger for the second tile, and the resulting tile expresses the synchronized behaviour of both. Vertical composition models the execution of a sequence of steps starting from an initial configuration. It corresponds to sequential composition of computations. The three compositions are illustrated in Figure 1.

Given a set of basic tiles, the associated tile logic is obtained by adding some canonical 'auxiliary' tiles and then closing by composition – horizontally, vertically, and in parallel – both auxiliary and basic tiles. As an example, auxiliary tiles may be introduced that accommodate isomorphic transformations of interfaces, yielding consistent rearrangements of configurations and observations.

Tile logic deals with algebraic structures on configurations that can be different from the ordinary, tree-like presentation of terms, as e.g., term graphs, partitions, preorders, relations, net processes. In fact, all these structures give rise to monoidal categories and, therefore, possess the two basic operations needed by tile configurations – the monoidal tensor product gives the parallel composition and the sequential composition of the category represents the application of a context to its arguments. In this paper we shall use (monoidal) categories in a very elementary way, i.e., just for representing terms and substitution diagrammatically as abstract arrows (from the arguments to the result). In particular we shall consider neither the categorical models of tile logic (expressed by suitable *monoidal double categories* [12,6]), nor the axiomatized proof terms decorating tile sequents. Varying the algebraic structure of configurations and observations tiles can model many different aspects of dynamic systems, ranging from synchronization of net transitions [7], to causal dependencies for located calculi and finitely branching approaches for name-passing calculi [11], to actor systems [21]. In addition, tile logic allows one to reason about terms with variables as Larsen and Xinxin's context systems [14], while SOS formats work for ground terms only.

Several formats for tiles have been defined where the structure of configurations is given by the term algebra over a signature. Namely, (1) the *monoidal tile format* [17], which has monoidal structures of both configurations and observations; (2) the *algebraic tile format* [12], which has a cartesian structure of configurations but only a monoidal structure of observations; and (3) the *term tile format* [6,4], which has cartesian structures of both configurations and observations. Although none of them ensures that tile bisimilarity is a congruence, by restricting these formats one can easily recover either the De Simone, or the positive GSOS, or the `zyft` format. Here, we shall focus only on the monoidal and term tile formats, showing that it is always possible to manage dynamic bisimulation via ordinary tile bisimulation by extending the vertical signature with a finite number of operators determined from the signature of configurations. In particular, for each operator f of the horizontal signature we shall add the observation $\tilde{f}$ and the tile



where *id* denotes identity in the appropriate category. Such a tile can then be applied to any configuration, embedding it in context f and producing effect $\tilde{f}$, which can now be observed in the ordinary bisimulation game. As we shall see, the congruence proof for bisimilarity in the enriched systems can be carried out as an abstract tile pasting. Moreover, such bisimilarity coincides with the dynamic bisimilarity on the original system.

The idea of allowing contexts as observations has been at the basis of the *promoted* `tyft/tyxt` format [1], designed for dealing with higher order languages. We think that such an extension can be carried out in a natural way in the abstract framework provided by tile logic.

Structure of the paper. In Section 1 we fix the notation, recall the most diffused rule formats for transition system specifications, and motivate dynamic bisimulation. In Section 2 we summarize the tile formats which we focus on. Section 3 introduces syntactical constraints on basic tiles that guarantee the 'bisimilarity as a congruence' property. Section 4 presents our main results: a finitary presentation of dynamic bisimulation via monoidal and term tile systems.

1 Bisimulation and SOS Formats

The notion of bisimulation dates back to the pioneering work of David Park and Robin Milner [18,22] on process algebras and provides the standard framework to express behavioural equivalences of complex dynamical systems.

Definition 1. *A* labeled transition system *(*LTS*) is a triple $L = (S, \Lambda, \rightarrow)$, where S is a set of* states*, Λ is a set of* labels*, and $\rightarrow$ is a ternary relation*

$$\rightarrow \subseteq S \times \Lambda \times S$$

We let $s, t, s', t' ...$ range over S and $a, b, c, ...$ range over Λ. For $\langle s, a, s' \rangle \in \rightarrow$ we use the notation $s \xrightarrow{a} s'$.

Definition 2. *For $L = (S, \Lambda, \rightarrow)$ a* LTS*, a* bisimulation *on L is a symmetric, reflexive relation $\sim \subseteq S \times S$ such that if $s \sim t$, then for any transition $s \xrightarrow{a} s'$ there exists a transition $t \xrightarrow{a} t'$ with $s' \sim t'$.*

We denote by $\simeq$ the largest bisimulation and call it *bisimilarity*, and we say that two states s and t are *bisimilar* whenever $s \simeq t$ or, equivalently, whenever there exists a bisimulation $\sim$ such that $s \sim t$.

In this paper we shall consider LTS whose states are terms over a given signature Σ. Although our results are easily extended to many-sorted signatures, for simplicity we focus on the one-sorted case. A *one-sorted signature* is a set Σ of *operators* together with an arity function $ar_\Sigma \colon \Sigma \rightarrow \mathbb{N}$ assigning to each operator the number of arguments it takes. The subset of Σ consisting of the operators of arity n is denoted by Σ_n. Operators in Σ_0 are called *constants*. We denote by $\mathbb{T}_\Sigma(X)$ the term algebra over Σ and variables in X (with X and Σ disjoint). We use $\mathbb{T}_\Sigma$ for $\mathbb{T}_\Sigma(\varnothing)$, the *term algebra* over Σ. For $t \in \mathbb{T}_\Sigma(X)$, we write $var(t)$ for the set of variables that appear in t. Term t is said *closed* or also *ground* if $var(t) = \varnothing$. We use a for a constant $a() \in \mathbb{T}_\Sigma(X)$.

A *substitution* is a mapping $\sigma \colon X \rightarrow \mathbb{T}_\Sigma(X)$. It is closed if each variable is mapped into a closed term. Substitutions extend to mappings from terms to terms as usual: $\sigma(t)$ is the term obtained by concurrently replacing all occurrences of variables x in t by $\sigma(x)$. The substitution mapping x_i to t_i for $i \in [1, n]$ is denoted by $[t_1/x_1, ..., t_n/x_n]$. Substitution σ' can be applied elementwise to substitution $\sigma = [t_1/x_1, ..., t_n/x_n]$ yielding the composed substitution

$$\sigma; \sigma' = \sigma'([t_1/x_1, ..., t_n/x_n]) = [\sigma'(t_1)/x_1, ..., \sigma'(t_n)/x_n].$$

A *context* $t = C[x_1, ., x_n]$ denotes a term in which at most the distinct variables $x_1, ..., x_n$ appear. The term $C[t_1, ..., t_n]$ is then obtained by applying the substitution $[t_1/x_1, ..., t_n/x_n]$ to $C[x_1, ., x_n]$. A context can therefore be regarded as a function from n terms to 1. Notice that the x_i may as well not appear in $C[x_1, ..., x_n]$. For example, the context $x_2[x_1, x_2, x_3]$ is a substitution from three arguments to one, which is the projection on the second argument.

A term is *linear* if each variable occurs at most once in it. Similarly, $\sigma = [t_1/x_1, ..., t_n/x_n]$ is *linear* if each t_i is linear and $var(t_i) \cap var(t_j) = \varnothing$ for $i \neq j$.

Substitutions and their composition $_;_$ form a (cartesian) category $\mathbf{Subs}_\Sigma$, with linear substitutions forming a monoidal subcategory. An alternative presentation of $\mathbf{Subs}_\Sigma$ can be obtained resorting to *algebraic theories*. An algebraic theory [15] is a cartesian category having 'underlined' natural numbers as objects. The free algebraic theory associated to a signature Σ is denoted by $\mathbf{Th}[\Sigma]$: the arrows from $\underline{m}$ to $\underline{n}$ are in a one-to-one correspondence with n-tuples of terms of the free Σ-algebra with (at most) m variables, and composition of arrows is term substitution. In particular, $\mathbf{Th}[\Sigma]$ is isomorphic to $\mathbf{Subs}_\Sigma$, and the arrows from $\underline{0}$ to $\underline{1}$ are in bijective correspondence with the closed terms over Σ. As a matter of notation, we assume a standard naming of the $\underline{m}$ input variables, namely $x_1, ..., x_m$. When composing two arrows $\vec{s}: \underline{m} \to \underline{k}$ and $\vec{t}: \underline{k} \to \underline{n}$, the resulting term $\vec{s}; \vec{t}$ is obtained by replacing each occurrence of x_i in $\vec{t}$ by the i-th term of the tuple $\vec{s}$, for $i \in [1, k]$. For example, constants a, b in Σ are arrows from $\underline{0}$ to $\underline{1}$, a binary operator $f(x_1, x_2)$ defines an arrow from $\underline{2}$ to $\underline{1}$, and the composition $\langle a, b\rangle; \langle f(x_1, x_2), x_1\rangle; f(x_2, x_1)$, where the angle brackets denote term tupling, yields the term $f(a, f(a, b))$, which is an arrow from $\underline{0}$ to $\underline{1}$. In fact,

$$\langle a, b\rangle; \langle f(x_1, x_2), x_1\rangle; f(x_2, x_1) = \langle f(a, b), a\rangle; f(x_2, x_1) = f(a, f(a, b))$$

Monoidal theories stay to algebraic theories as linear substitutions stay to generic substitutions. More precisely, in monoidal theories variables can be neither duplicated (as e.g. in $f(x_1, x_1)$) nor projected. Even though terms are formally annotated with the variables on which they are built, when no confusion can arise, we avoid such annotations, and also the use of angle brackets.

LTS defined over closed terms of a given signature Σ and label alphabet Λ can be conveniently specified as collections of inductive proof rules, called *transition system specifications*. A *transition rule* α has the form

$$\alpha : \frac{s_1 \xrightarrow{a_1} t_1 \ ... \ s_n \xrightarrow{a_n} t_n}{s \xrightarrow{a} t}$$

where the s_i, t_i, s and t range over $\mathbb{T}_\Sigma(X)$ and the a_i, a range over Λ. Transitions in the upper part of the rule are called *premises*, the one in the lower part *conclusion*. The rule α is closed if it does not contain variables. A *transition system specification* (TSS) is a set of transition rules.

A proof of a closed *transition rule* $H/s \xrightarrow{a} t$ with H a set of (closed) premises, is a well-founded, upwardly branching tree whose nodes are labeled by closed transitions, the root is labeled by $s \xrightarrow{a} t$ and if H_r is the set of labels for the

nodes above a node r with label $s_r \xrightarrow{a_r} t_r$ then either $H_r/s_r \xrightarrow{a_r} t_r$ is a closed rule that can be obtained as an instance of a rule in the TSS, or $s_r \xrightarrow{a_r} t_r \in H$ and $H_r = \varnothing$. A proof of a closed *transition* $s \xrightarrow{a} t$ is a proof of $\varnothing/s \xrightarrow{a} t$. The LTS *associated* to a TSS consists of the set of provable closed transitions.

Among the formats that guarantee the fundamental property of 'bisimilarity as congruence' on the associated LTS we shall in the following recall the *De Simone*, the GSOS and the `tyft` formats. We remind that an equivalence relation $\mathcal{R}$ over $\mathbb{T}_\Sigma$ is a *congruence* if it respects the algebraic structure of states given by Σ, i.e. if for all $f \in \Sigma$

$$s_i \,\mathcal{R}\, t_i, \text{ for } i \in [1, ar(f)], \quad \text{implies} \quad f(s_1, ..., s_{ar(f)}) \,\mathcal{R}\, f(t_1, ..., t_{ar(f)}).$$

Definition 3. *Let Σ be a signature and Λ an alphabet of observations. A transition rule is in* De Simone format *if it has the form*

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{f(x_1, ..., x_n) \xrightarrow{a} t}$$

where the a_i and a are labels in Λ, f is a n-ary operator, $I \subseteq \{1, ..n\}$ and the variables x_i and y_i are all distinct and form the set V. Moreover, the target $t \in \mathbb{T}_\Sigma(V)$ does not contain x_i for $i \in I$ and is linear. A TSS *is in De Simone format if all its rule are in such form.*

Observe that the source of the conclusion of a rule in De Simone format consists of a *single function symbol*. This fact is crucial in the proof that bisimilarity is a congruence for TSS specified in De Simone format.

The *positive* GSOS *format* [3] extends De Simone rules in several ways: (1) multiple testings of the same argument (the x_i) are allowed in the premises; (2) tested arguments can appear in the target t of the conclusion; (3) the target t of the conclusion can be a non-linear context (variables can be used more than once). The `tyft` *format* [13] further generalizes the GSOS, by allowing generic terms t_i as sources of the transitions in the premises. Notice however that the main restriction about the source of the conclusion still persists: allowing conclusions of the form $C[x_1, ..., x_n] \xrightarrow{a} t$ could compromise the 'bisimilarity as congruence' property as the following example illustrates.

Example 1. Consider a process algebra over the signature $\Sigma = \{nil, \alpha._-, \bar{\alpha}._-, _ \mid _\}$ with α ranging over a suitable set of channels A, where *nil* is the empty agent, $\alpha._-$ and $\bar{\alpha}._-$ are two complementary unary operator for action prefix (on the channel α) and $_ \mid _$ is a binary parallel composition operator. If we consider the TTS consisting of the axiom $\lambda.x_1 \mid \bar{\lambda}.x_2 \xrightarrow{\tau} x_1 \mid x_2$ plus the usual rules that propagate the τ through the $_ \mid _$ operator (asynchronously), then it is obvious that $\alpha.nil \simeq \bar{\alpha}.nil$. However, if put in the context $x_1 \mid \bar{\alpha}.nil$, then $\alpha.nil \mid \bar{\alpha}.nil \xrightarrow{\tau} nil \mid nil$, while $\bar{\alpha}.nil \mid \bar{\alpha}.nil$ cannot move. Hence $\alpha.nil \mid \bar{\alpha}.nil \not\simeq \bar{\alpha}.nil \mid \bar{\alpha}.nil$.

Dynamic bisimulation has been defined to reproduce the effect of run-time reconfiguration in open ended systems. It extends the ordinary bisimulation-game by allowing moves that put the states under comparison in the same context.

Definition 4. *Given a* LTS $L = (\mathbb{T}_\Sigma, \Lambda, \rightarrow)$*, a* dynamic bisimulation *on L is a symmetric, reflexive relation $\sim_{\mathrm{d}} \subseteq \mathbb{T}_\Sigma \times \mathbb{T}_\Sigma$ such that if $s \sim_{\mathrm{d}} t$ then for any unary context $C[_]$ (including the identity) and transition $C[s] \xrightarrow{a} s'$ there exists a transition $C[t] \xrightarrow{a} t'$ with $s' \sim_{\mathrm{d}} t'$.*

Two states s and t are *dynamic bisimilar*, written $s \simeq_{\mathrm{d}} t$, if there exists a dynamic bisimulation $\sim_{\mathrm{d}}$ such that $s \sim_{\mathrm{d}} t$. Thus, w.r.t. Example 1, we have, e.g., $\alpha.nil \not\simeq_{\mathrm{d}} \bar{\alpha}.nil$. Dynamic bisimilarity is the *coarsest congruence* which is also a *bisimulation*. Note that context moves cannot be 'observed': they are part of the game but not of the LTS. Even if not remarked in [20], dynamic bisimulation can however be recasted in ordinary bisimulation over an extended system. Observe that the dynamic extension is an *infinitary* construction on the LTS and is not expressed at the level of the TSS, i.e., it remains at the '*meta-level*'.

Definition 5. *Given a* LTS $L = (\mathbb{T}_\Sigma, \Lambda, \rightarrow)$*, its* dynamic extension $\widehat{L}$ *is the* LTS $(\mathbb{T}_\Sigma, \Lambda, \rightarrow \cup \rightrightarrows)$*, where* $s \xrightarrow{C[_]} C[s]$ *for all $s \in \mathbb{T}_\Sigma$ and unary contexts $C[_]$.*

Proposition 1. *$s \simeq_{\mathrm{d}} t$ in L iff $s \simeq t$ in $\widehat{L}$.*

The proof of Proposition 1 relies on the fact that if s makes a move $C[_]$, then t can always simulate such move in a unique way.

2 Tile Formats

A *tiles system* is a tuple $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ where $\mathcal{H}$ and $\mathcal{V}$ are monoidal categories with the same set of objects $O_{\mathcal{H}} = O_{\mathcal{V}}$, N is the set of rule names and $R\colon N \rightarrow \mathcal{H} \times \mathcal{V} \times \mathcal{V} \times \mathcal{H}$ is a function such that for all $\alpha \in N$, if $R(\alpha) = \langle s, a, b, t \rangle$, then we have $s\colon x \rightarrow y$, $a\colon x \rightarrow z$, $b\colon y \rightarrow w$, and $t\colon z \rightarrow w$ for suitable objects x, y, z and w. We shall write such rule either as the sequent $\alpha\colon s \xrightarrow[b]{a} t$, or as the tile

$$\begin{array}{ccc} x & \xrightarrow{\ s\ } & y \\ {\scriptstyle a}\downarrow & \alpha & \downarrow{\scriptstyle b} \\ z & \xrightarrow[\ t\ ]{} & w \end{array}$$

thus making explicit the source and target of each arrow. The category $\mathcal{H}$ is called *horizontal* and its arrows *configurations*. The category $\mathcal{V}$ is called *vertical* and its arrows *observations*. The objects of $\mathcal{H}$ and $\mathcal{V}$ are called *interfaces*.

Starting from the basic tiles $R(\alpha)$ of the system, more complex tiles can be constructed via horizontal, vertical and parallel composition. Moreover, the horizontal and vertical identities are always added to the system and composed with the basic tiles. All this is illustrated in Figure 2. Depending on the chosen tile format, $\mathcal{H}$ and $\mathcal{V}$ must satisfy certain constraints and suitable *auxiliary tiles* are added and composed with basic tiles and identities in all the possible ways. The set of resulting tiles (called *flat sequents*) define the *flat tile logic* associated to $\mathcal{R}$. We say that $s \xrightarrow[b]{a} t$ is *entailed* by the logic, written $\mathcal{R} \vdash s \xrightarrow[b]{a} t$, if the sequent $s \xrightarrow[b]{a} t$ can be expressed as the composition of basic and auxiliary tiles.

$$\frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[c]{b} f}{s;h \xrightarrow[c]{a} t;f} \qquad \frac{s \xrightarrow[b]{a} t \quad t \xrightarrow[d]{c} h}{s \xrightarrow[b;d]{a;c} h} \qquad \frac{s \xrightarrow[b]{a} t \quad h \xrightarrow[d]{c} f}{s \otimes h \xrightarrow[b\otimes d]{a\otimes c} t \otimes f}$$

$$\frac{t\colon x \to y \in \mathcal{H}}{t \xrightarrow[y]{x} t} \qquad \frac{a\colon x \to z \in \mathcal{V}}{x \xrightarrow[a]{a} z}$$

Figure 2.

Definition 6. *Let $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ be a tile system. A symmetric relation $\sim_{\mathrm{t}}$ on configurations is called* tile bisimulation *if whenever $s \sim_{\mathrm{t}} t$ and $\mathcal{R} \vdash s \xrightarrow[b]{a} s'$, then there exists t' such that $\mathcal{R} \vdash t \xrightarrow[b]{a} t'$ and $s' \sim_{\mathrm{t}} t'$.*

The maximal tile bisimulation is denoted by $\simeq_{\mathrm{t}}$, and two configurations s and t are said to be *tile bisimilar* if $s \simeq_{\mathrm{t}} t$.

We are particularly interested in considering tiles systems where the monoidal categories of configurations and observations are freely generated from suitable horizontal and vertical signatures, respectively, i.e., they are categories of *substitutions*, as discussed in the previous section.

The tile format proposed in the original presentation of tiles [12] is the so-called *algebraic tile format* that recollected the perspective of ordinary TSS: configurations are terms over a certain signature, and observations are the arrows of the monoidal category freely generated by certain labels (regarded as unary operators). Auxiliary tiles lift the horizontal cartesian structure to the horizontal composition of tiles. In the algebraic tile format basic tiles have the form:

$$\begin{array}{ccc} \underline{n} & \xrightarrow{\quad s \quad} & \underline{1} \\ {\scriptstyle a_1\otimes\cdots\otimes a_n}\downarrow & & \downarrow{\scriptstyle a} \\ \underline{n} & \xrightarrow[\quad t \quad]{} & \underline{1} \end{array}$$

where the a_i and a can be either labels (viewed as arrows from $\underline{1}$ to $\underline{1}$) or identities and $s, t \in \mathbb{T}_{\Sigma}(\{x_1, ..., x_n\})$. The idea is that each interface represents an ordered sequence of variables; therefore each variable is completely identified by its position in the tuple, and a standard naming $x_1, ..., x_n$ of the variables can be assumed for all interfaces. A typical auxiliary tile for the algebraic format is

$$\begin{array}{ccc} \underline{1} & \xrightarrow{\quad \langle x_1, x_1 \rangle \quad} & \underline{2} \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a\otimes a} \\ \underline{1} & \xrightarrow[\quad \langle x_1, x_1 \rangle \quad]{} & \underline{2} \end{array}$$

that duplicates the observation a (trigger of the tile) propagating it to two instances of the unique variable in the initial interface. We refer to [12] for more details. The algebraic tile format corresponds to SOS rules of the form

$$\frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{C[x_1, ..., x_n] \xrightarrow{a} D[y_1, ..., y_n]}$$

where $I \subseteq \{1, ..., n\}$, $C[x_1, ..., x_n]$ and $D[y_1, ..., y_n]$ are contexts (corresponding to s and t in the tile), and all the y_i and x_i are different if $i \in I$, but $y_i = x_i$ otherwise. The correspondence follows since for all closed terms s and t and for any label a, $\mathcal{R} \vdash s \xrightarrow[a]{id_0} t$ if and only if the LTS associated to the SOS specification includes the transition $t \xrightarrow{a} s$.

The algebraic tile format is not uniform in the two dimensions, since $\mathcal{H}$ is cartesian, whereas $\mathcal{V}$ is only monoidal (non symmetric). Since our idea is to observe contexts by replicating (part of) the horizontal structure in the vertical dimension, we prefer (1) to renounce to the cartesian structure altogether, resorting to the simpler *monoidal tile format* [17] where only *linear* contexts are allowed, or (2) consider the more general *term tile format* [4], where also $\mathcal{V}$ is cartesian. Notice that monoidal theories suffice for expressing all closed terms, even though, as explained in [5], the term tile format is more expressive.

Since in all tile formats the categories of configurations and observations are *freely generated* by the horizontal signature Σ and by (the signature associated to) the set of labels Λ, monoidal/algebraic/term tile systems are usually represented as tuples of the form $\mathcal{R} = (\Sigma, \Lambda, N, R)$.

According to the term tile format each basic tile has the form:

$$\begin{array}{ccc} \underline{n} & \xrightarrow{\vec{h}} & \underline{m} \\ {\scriptstyle \vec{v}}\downarrow & & \downarrow{\scriptstyle u} \\ \underline{k} & \xrightarrow[g]{} & \underline{1} \end{array}$$

with $\vec{h} \in \mathbb{T}_{\Sigma^{\mathrm{H}}}(X_n)^m$, $g \in \mathbb{T}_{\Sigma^{\mathrm{H}}}(X_k)$, $\vec{v} \in \mathbb{T}_{\Sigma^{\mathrm{V}}}(X_n)^k$, and $u \in \mathbb{T}_{\Sigma^{\mathrm{V}}}(X_m)$, where $X_i = \{x_1, ..., x_i\}$ is a chosen set of variables, Σ^{H} is the horizontal signature of configurations and Σ^{V} is the vertical signature of observations. Of course, if Λ contains only elementary actions, regarded as unary operators, then $m = 1$. We present tiles more concisely as logic sequents $n \triangleleft \vec{h} \xrightarrow[u]{\vec{v}} g$, where the number of variables in the 'upper-left' corner of the tile is made explicit (the values m and k can be recovered from the lengths of $\vec{h}$ and $\vec{v}$). Again, a standard naming for the variables in the interfaces is assumed. For example, if the variable x_i appears in the effect u of the above rule, then the effect u depends on the ith component h_i of the initial configuration. Analogously for the remaining connections. As already remarked, the same variable x_i denotes the ith element of different interfaces when used in each of the four border-arrows of the tile (as a matter of fact, only the occurrences of x_i in $\vec{h}$ and in $\vec{v}$ denote the same element of the initial input interface $\underline{n}$).

Auxiliary tiles for term tile systems consist of all term tiles $n \triangleleft h \xrightarrow[u]{v} g$ such that h, g, u and v are terms over the empty signature – and therefore also terms of $\mathbb{T}_{\Sigma^{\mathrm{V}}}(X)$ and $\mathbb{T}_{\Sigma^{\mathrm{H}}}(X)$ – such that $h; u = v; g$, i.e., all tiles that perform consistent rearrangements of a generic interface in the two dimensions. A typical auxiliary term tile is $1 \triangleleft x_1 \xrightarrow[x_1, x_1]{x_1} x_1, x_1$ that consistently duplicates the unary interface. Observe that term tile format *extends* the positive GSOS format.

An interesting question concerns suitable restrictions of the monoidal and term tile formats such that tile bisimilarity yields a congruence. Two main properties have been investigated in the literature for obtaining tile bisimilarity congruences: the *basic source* and the *tile decomposition*. Tile decomposition has a completely abstract formulation that applies to all tile systems.

Definition 7. *A tile system* $\mathcal{R} = (\mathcal{H}, \mathcal{V}, N, R)$ *enjoys the* decomposition property *if for all arrows* $s: x \to y \in \mathcal{H}$ *and for all sequents* $s \xrightarrow[b]{a} t$ *entailed by* $\mathcal{R}$

- *if* $s = s_1; s_2$ *then there exists* $c \in \mathcal{V}$ *and* $t_1, t_2 \in \mathcal{H}$ *such that* $\mathcal{R} \vdash s_1 \xrightarrow[c]{a} t_1$, $\mathcal{R} \vdash s_2 \xrightarrow[b]{c} t_2$ *and* $t = t_1; t_2$;
- *if* $s = s_1 \otimes s_2$ *then there exists* $a_1, a_2, b_1, b_2 \in \mathcal{V}$ *and* $t_1, t_2 \in \mathcal{H}$ *such that* $\mathcal{R} \vdash s_1 \xrightarrow[b_1]{a_1} t_1$, $\mathcal{R} \vdash s_2 \xrightarrow[b_2]{a_2} t_2$, $a = a_1 \otimes a_2$, $b = b_1 \otimes b_2$ *and* $t = t_1 \otimes t_2$;

Proposition 2. *If* $\mathcal{R}$ *enjoys the decomposition property, then tile bisimilarity is a congruence (w.r.t. the operations of the horizontal category, i.e., sequential and parallel composition).*

Even though we did not address it explicitly, all the definitions we have given for tiles apply also to the case of term algebras modulo structural axioms (e.g., associativity and commutativity of parallel composition in CCS) and all our results can be immediately extended.

3 Basic Source

The results in this section mildly extend those of [12]. Since in this paper we consider equivalences on closed terms, we refine the notion of tile bisimulation to *ground tile bisimulation*.

Definition 8. *Let* $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$ *be a monoidal (resp. term) tile system. A symmetric relation* $\sim_{\mathrm{g}}$ *on closed configurations (i.e., elements of* $\mathbb{T}_{\Sigma^{\mathrm{H}}}$*) is called* ground tile bisimulation *if whenever* $s \sim_{\mathrm{g}} t$ *and* $\mathcal{R} \vdash s \xrightarrow[a]{id_0} s'$*, then there exists* t' *such that* $\mathcal{R} \vdash t \xrightarrow[a]{id_0} t'$ *and* $s' \sim_{\mathrm{g}} t'$.

Ground tile bisimulation is the exact counterpart of ordinary bisimulation for LTS. It differs from tile bisimulation in that is not defined on contexts. (Since ground terms need no trigger, ground bisimulation tests only the effects they can produce.) The maximal ground tile bisimulation is denoted by $\simeq_{\mathrm{g}}$, and two closed configurations s and t are said to be *ground tile bisimilar* if $s \simeq_{\mathrm{g}} t$. In fact we are interested in the LTS associated to tile systems.

Definition 9. *For* $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$ *a monoidal/term tile system, the* LTS *associated to* $\mathcal{R}$ *is* $L_{\mathcal{R}} = (\mathbb{T}_{\Sigma^{\mathrm{H}}}, \mathbb{T}_{\Sigma^{\mathrm{V}}}(\{x_1\}), \to)$ *where* $s \xrightarrow{a} t$ *iff* $\mathcal{R} \vdash s \xrightarrow[a]{id_0} t$.

The decomposition property can be refined and related to the 'tile bisimilarity as congruence' property.

Definition 10. *A term (resp. monoidal) tile system $\mathcal{R}$ enjoys the* ground decomposition property *if for any $s \in \mathbb{T}_{\Sigma^{\mathrm{H}}}$ and any sequent $\mathcal{R} \vdash C[s_1] \xrightarrow[b]{id_0} t$ with C a unary (resp. linear unary) context and s_1 a ground term such that $s = C[s_1]$, then there exists an observation c, a ground term t_1 and a (resp. linear) context D such that $\mathcal{R} \vdash s_1 \xrightarrow[c]{id_0} t_1$ and $\mathcal{R} \vdash C[x_1] \xrightarrow[b]{c} D[x_1]$, with $t = D[t_1]$.*

Theorem 1. *Let $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$ be a term tile system. The ground decomposition property implies that ground tile bisimilarity on $\mathcal{R}$ is a congruence.*

Proof. Standard. Define the congruence $\hat{\simeq}_{\mathrm{g}}$ as the minimal relation such that if $s \simeq_{\mathrm{g}} t$ and $C[x_1]$ is a unary context then $C[s] \hat{\simeq}_{\mathrm{g}} C[t]$. Obviously $\simeq_{\mathrm{g}} \subseteq \hat{\simeq}_{\mathrm{g}}$. We then show that $\hat{\simeq}_{\mathrm{g}}$ is a ground tile bisimulation and, therefore, coincides with ground tile bisimilarity. In fact, let $C[s] \simeq_{\mathrm{g}} C[t]$ for $s, t \in \mathbb{T}_{\Sigma^{\mathrm{H}}}$ with $s \simeq_{\mathrm{g}} t$ and $C[x_1]$ a unary context. By ground decomposition we have that if $\mathcal{R} \vdash C[s] \xrightarrow[a]{id_0} s'$, there exist $s_1 \in \mathbb{T}_{\Sigma^{\mathrm{H}}}$, an observation b and a unary context $D[x_1]$ such that $\mathcal{R} \vdash s \xrightarrow[b]{id_0} s_1$, $\mathcal{R} \vdash C[x_1] \xrightarrow[a]{b} D[x_1]$ and $s' = D[s_1]$. Since $s \simeq_{\mathrm{g}} t$ then $\mathcal{R} \vdash t \xrightarrow[b]{id_0} t_1$ for some $t_1 \in \mathbb{T}_{\Sigma^{\mathrm{H}}}$ with $s_1 \simeq_{\mathrm{g}} t_1$. By horizontal composition of tiles we then have $\mathcal{R} \vdash C[t] \xrightarrow[a]{id_0} D[t_1]$. By definition of $\hat{\simeq}_{\mathrm{g}}$ we have that $D[s_1] \hat{\simeq}_{\mathrm{g}} D[t_1]$. □

Theorem 2. *Given a monoidal tile system $\mathcal{R} = (\Sigma, \Lambda, N, R)$, the ground decomposition property implies that ground tile bisimilarity is a congruence.*

Proof. Similar to the proof of Theorem 1, but requires $\hat{\simeq}_{\mathrm{g}}$ to be the minimal relation such that if $s \simeq_{\mathrm{g}} t$ and $C[x_1]$ is a *linear* (rather than generic) unary context then $C[s] \hat{\simeq}_{\mathrm{g}} C[t]$. Observe that the congruence property then holds for generic contexts. In fact, if $D[_]$ is not linear, let $D'[x_1, x_2, ..., x_n]$ be the linear context obtained from $D[_]$ by replacing each occurrence of the hole '_' by a different variable x_i. Then given any two closed terms s and t we have

$$D[s] = D'[\underbrace{s, s, ..., s}_{n}] \hat{\simeq}_{\mathrm{g}} D'[t, \underbrace{s, ..., s}_{n-1}] \hat{\simeq}_{\mathrm{g}} D'[t, t, \underbrace{s, ..., s}_{n-2}] \hat{\simeq}_{\mathrm{g}} \cdots \hat{\simeq}_{\mathrm{g}} D'[\underbrace{t, t, ..., t}_{n}] = D[t]$$

since all contexts $D'[_, s, ..., s]$, $D'[t, _, s, ..., s]$, ..., $D'[t, ..., t, _]$ are linear. □

The ground decomposition property can be enforced by syntactical constraints on basic tiles. A tile system verifies the *basic source property* if the initial configuration of each basic tile consists of a *single operator*, rather than a generic context.

Proposition 3. *If a monoidal (resp. term) tile system $\mathcal{R}$ enjoys the basic source property, then the ground tile bisimilarity on $\mathcal{R}$ is a congruence.*

The proof of Proposition 3 consists of two steps: we first prove that basic source implies *ground tile decomposition* and then we conclude by exploiting Theorems 1 and 2. Although Theorems 1 and 2 hold also when structural axioms are imposed on configurations, this is not necessarily the case for Proposition 3, as the first part of the argument above may fail. We remark that for the monoidal tile format the proof coincides with that for the De Simone format, since the two formats are essentially the same.

4 Dynamic Tile Bisimulation

When the basic source property is not satisfied we are likely to have bisimilarities that are not congruences. This consideration applies to all the most diffused formats, unless the specification contains enough rules to distinguish context dependent redexes. For example, Corradini and Heckel in joint work with the second author [8] suggested one may deal with this situation via a closure operation on the TSS rules. And Sewell in [24] when passing from reduction systems to transition systems performs such a closure by adding a transition labeled with C for each state s and context C which s can react with in order to perform a reduction. However, such closures are expressed at a meta-level, as they are not handled by adding rules to the original specification. Therefore, from a different perspective, dynamic bisimilarity is more satisfactory, since it allows for a concise definition of the congruence one is looking for.

Tiles have the expressive power to reconcile finitary system specifications and dynamic bisimulation within the same perspective, i.e., by adding suitable auxiliary tiles. Moreover, the closure w.r.t. all contexts can be expressed simply by adding just as many basic tiles as the operators in the signature. Hence, if the signature is finite, so are the additional auxiliary tiles needed.

Definition 11. *Given a term (resp. monoidal) tile system* $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$ *its* dynamic extension $\widehat{\mathcal{R}}$ *is obtained by adding for all* n *and for any operator* $f \in \Sigma^{\mathrm{H}}_n$ *the auxiliary operator* $\tilde{f}$ *to* Σ^{V}_n *and the following auxiliary tiles.*

$$\begin{array}{ccc} \underline{n} & \xrightarrow{x_1,.,x_n} & \underline{n} \\ {\scriptstyle x_1,.,x_n}\downarrow & & \downarrow{\scriptstyle \tilde{f}(x_1,.,x_n)} \\ \underline{n} & \xrightarrow[f(x_1,.,x_n)]{} & \underline{1} \end{array} \qquad \begin{array}{ccc} \underline{n} & \xrightarrow{f(x_1,.,x_n)} & \underline{1} \\ {\scriptstyle \tilde{f}(x_1,.,x_n)}\downarrow & & \downarrow{\scriptstyle x_1} \\ \underline{1} & \xrightarrow[x_1]{} & \underline{1} \end{array}$$

For t a generic horizontal context, we let $\tilde{t}$ denote the corresponding vertical context on the extended signature, which is obtained by replacing each operator f that appears in t by its vertical counterpart $\tilde{f}$, leaving variables unchanged. For the proof of the main theorem we need the following technical lemmas that require some acquaintance with the term tile format. A corresponding lemma can be proved for the monoidal tile format, by considering linear contexts only.

Lemma 1. *Given a term tile system* $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$*, for each context* $t\colon \underline{n} \to \underline{1}$ *we have* $\widehat{\mathcal{R}} \vdash n \triangleleft x_1, ..., x_n \xrightarrow[\tilde{t}]{x_1,...,x_n} t$.

Proof. The proof proceeds by induction on the (maximum) depth m of the tree-like representation of t. The base case $m = 0$ is trivial. If $m > 1$ then $t = f(t_1, ..., t_k)$, where f is a k-ary operator of Σ^{H} and $t_1, ..., t_k$ are context with (maximum) depth strictly less than m. Then we can apply the inductive hypothesis to conclude that $n \triangleleft x_1, ..., x_n \xrightarrow[\tilde{t_i}]{x_1,...,x_n} t_i$ for $i \in [1, k]$. Composing in parallel such sequents we get $k \cdot n \triangleleft x_1, ..., x_{k \cdot n} \xrightarrow[\tilde{s_1},...,\tilde{s_k}]{x_1,...,x_{k \cdot n}} s_1, ..., s_k$, where $s_i = t_i[x_{n \cdot (i-1)+1}/x_1, \ldots, x_{n \cdot (i-1)+n}/x_n]$, i.e., the s_i are the t_i with the variables suitably renamed according to the initial input interface. Then we can horizontally compose with the auxiliary tile of term tile systems that makes k copies of n inputs $n \triangleleft x_1, ., x_n, ..., x_1, ., x_n \xrightarrow[x_1,...,x_{k \cdot n}]{x_1,.,x_n,...,x_1,.,x_n} x_1, ., x_{k \cdot n}$, obtaining the sequent $\alpha = n \triangleleft x_1, ..., x_n \xrightarrow[\tilde{t_1},...,\tilde{t_k}]{x_1,...,x_n} t_1, ..., t_k$. Finally we can compose it with the auxiliary sequent of the extended system $\beta = k \triangleleft x_1, ..., x_k \xrightarrow[\tilde{f}(x_1,...,x_k)]{x_1,...,x_k} f(x_1, ..., x_k)$ as in

$$\begin{array}{ccc} \cdot & \longrightarrow & \cdot \\ \downarrow & \alpha & \downarrow \\ \cdot & \longrightarrow & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & \longrightarrow & \cdot \\ \downarrow & \gamma & \downarrow \\ \cdot & \longrightarrow & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & \longrightarrow & \cdot \\ \downarrow & \delta & \downarrow \\ \cdot & \longrightarrow & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & \longrightarrow & \cdot \\ \downarrow & \beta & \downarrow \\ \cdot & \longrightarrow & \cdot \end{array}$$

where γ is the horizontal identity for the effect of α and δ is the vertical identity for the final configuration of α. The composition yields the sequent $n \triangleleft x_1, ..., x_n \xrightarrow[\tilde{f}(\tilde{t_1},...,\tilde{t_k})]{x_1,...,x_n} f(t_1, ..., t_k) = n \triangleleft x_1, ..., x_n \xrightarrow[\tilde{t}]{x_1,...,x_n} t$ and concludes the proof. □

A similar argument shows the following lemma.

Lemma 2. *Given a term tile system $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$, for each context $t: \underline{n} \to \underline{1}$ we have $\widehat{\mathcal{R}} \vdash n \triangleleft t \xrightarrow[x_1]{\tilde{t}} x_1$.*

Theorem 3. *Let $\mathcal{R} = (\Sigma^{\mathrm{H}}, \Sigma^{\mathrm{V}}, N, R)$ be a term tile system. The ground tile bisimilarity defined on $\widehat{\mathcal{R}}$ defines a congruence for $\mathcal{R}$.*

Proof. First notice that the auxiliary tiles do not influence the definition of ground tile bisimilarity, which deals only with null triggers. Then, we prove that $\widehat{\mathcal{R}}$ enjoys the ground decomposition property from which we get the expected 'bisimilarity as a congruence' property. In fact, given a generic sequent $\alpha: C[s] \xrightarrow[a]{id_0} t$ entailed by $\widehat{\mathcal{R}}$, we can always construct two tiles with source s and $C[x_1]$ respectively that decompose α, as illustrated in Figure 3. □

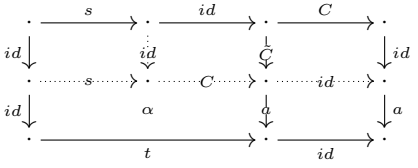


Figure 3.

Theorem 4. *Ground tile bisimilarity on $\widehat{\mathcal{R}}$, denoted by $\simeq_{\hat{g}}$, coincides with the dynamic bisimilarity on $L_{\mathcal{R}}$.*

Proof. We must show that $L_{\widehat{\mathcal{R}}} = \widehat{L}_{\mathcal{R}}$. The inclusion $L_{\widehat{\mathcal{R}}} \supseteq \widehat{L}_{\mathcal{R}}$ follows directly from the technical lemmas above, whilst the inclusion $L_{\widehat{\mathcal{R}}} \subseteq \widehat{L}_{\mathcal{R}}$ is more involved. The key point is showing that if an auxiliary 'context' tile gives rise to a new transitions, its label $\tilde{f}$ appears manifestly, i.e., the use of auxiliary tiles cannot be 'hidden' inside the proof to originate unexpected reactions. To show this, let $\mathcal{R} \vdash s \xrightarrow[a]{id_0} t$ and suppose that the proof of $s \xrightarrow[a]{id_0} t$ contains the auxiliary tile $\alpha = \vec{x} \xrightarrow[\tilde{f}(\vec{x})]{\vec{x}} f(\vec{x})$, for some operator f. We proceed by induction on the number k of such auxiliary tiles and then by case analysis. If $k = 0$ then $s \xrightarrow{a} t \in L_{\mathcal{R}} \subseteq \widehat{L}_{\mathcal{R}}$. If $k > 1$ then we take one such auxiliary tile α in the proof and examine the following three cases: (1) the effect of α is propagated to the final effect $a = A[\tilde{f}(\vec{a})]$ and thus can be observed; (2) the tile α is horizontally composed with $\beta = f(\vec{x}) \xrightarrow[x_1]{\tilde{f}(\vec{x})} x_1$ and thus does not appear in a; (3) the effect $\tilde{f}$ is vertically composed with other effects that override it. In case (1), α corresponds to a context move in $\widehat{L}_{\mathcal{R}}$. In case (2), the composition of α with β yields the vertical identity on f which is of course entailed in $\mathcal{R}$. Finally, in case (3), the effect $\tilde{f}$ can only be overridden because of structural axioms on observations, and in particular those involving projections. But then it can be shown that if projections are used that throw $\tilde{f}$ away, then also the result of applying the context f in the intermediate state is thrown away in the proof. Therefore, we can always reduce to a proof with $k - 1$ auxiliary tiles for adding contexts and conclude the proof by inductive hypothesis. □

For monoidal tile systems the proof simplifies considerably, since case (3) cannot occur. We remark that if observations are subject to structural axioms (e.g., $b; a = a$ for all observations b), then such axioms *must not* be extended to the $\tilde{f}$, otherwise the case (3) of the proof could be compromised.

Example 2. Let us take again the process algebra of Example 1, with $_ \mid _$ associative (but neither commutative nor with unit). Then $\alpha.nil \mid \bar{\alpha}.nil \simeq_{\mathrm{g}} \beta.nil \mid \bar{\beta}.nil \not\simeq_{\hat{\mathrm{g}}} \alpha.nil \mid \bar{\alpha}.nil$. While, e.g., $t_\alpha \simeq_{\mathrm{g}} t_\beta \simeq_{\hat{\mathrm{g}}} t_\alpha$ for $t_\lambda = nil \mid \lambda.nil \mid \bar{\lambda}.nil \mid nil$.

Concluding Remarks

We have proposed tile logic as a compositional framework suitable to deal with open ended systems, dynamic bisimulation and structural axioms on states. Such characteristics follows naturally from the abstract 'geometrical' concepts which tile configurations and observations are based on. In particular, the winning feature, is the possibility of exploiting the analogy between horizontal and vertical arrows, making observations out of contexts. Moreover, dynamic bisimulation is handled via a finitary enrichment of the specification and the congruence proof has a simple pictorial representation that exploits ground decomposition.

Following the lines suggested in this paper, one could limit run-time reconfiguration either to a sub-class of contexts or to a sub-class of configurations. Although we have not discussed the issue here, tile logic can deal with trace semantics as well.

References

1. K. Bernstein. A congruence theorem for structured operational semantics of higher-order languages. In *Proc. 13th LICS*, IEEE Press, 1998.
2. G. Berry and G. Boudol. The chemical abstract machine. *Theoret. Comput. Sci.*, 96(1):217–248, 1992.
3. B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can't be traced. *Journal of the ACM*, 42(1):232–268, 1995.
4. R. Bruni, J. Meseguer, and U. Montanari. Process and term tile logic. Technical Report SRI-CSL-98-06. SRI International, 1998.
5. R. Bruni, J. Meseguer, and U. Montanari. Executable tile specifications for process calculi. In *Proc. FASE'99*, vol. 1577 of *LNCS*, pages 60–76, Springer, 1999.
6. R. Bruni, J. Meseguer, and U. Montanari. Symmetric monoidal and cartesian double categories as a semantic framework for tile logic. *Mathematical Structures in Computer Science*, 2000. To appear.
7. R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Information and Computation*, 156:46–89, 2000.
8. A. Corradini, R. Heckel, and U. Montanari. From SOS specifications to structured coalgebras: how to make bisimulation a congruence. In *Proc. CMCS'99*, vol. 19 of *Elect. Notes in Th. Comput. Sci.*, Elsevier Science, 1999.
9. A. Corradini and U. Montanari. An algebraic semantics for structured transition systems and its application to logic programs. *Th. Comput. Sci.*, 103:51–106, 1992.
10. R. De Simone. Higher level synchronizing devices in MEIJE–SCCS. *Theoret. Comput. Sci.*, 37:245–267, 1985.
11. G.L. Ferrari and U. Montanari. Tile formats for located and mobile systems. *Information and Computation*, 156:173–235, 2000.
12. F. Gadducci and U. Montanari. The tile model. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, MIT Press, 2000. To appear.
13. J.F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100:202–260, 1992.
14. K.G. Larsen and L. Xinxin. Compositionality through an operational semantics of contexts. In *Proc. ICALP'90*, vol. 443 of *LNCS*, pages 526–539, Springer, 1990.

15. F.W. Lawvere. Functorial semantics of algebraic theories. *Proc. National Academy of Science*, 50:869–872, 1963.
16. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.
17. J. Meseguer and U. Montanari. Mapping tile logic into rewriting logic. In *Proc. WADT'97*, vol. 1376 of *Lect. Notes in Comput. Sci.*, pages 62–91, Springer, 1998.
18. R. Milner. *A Calculus of Communicating Systems*, vol. 92 of *LNCS* Springer, 1980.
19. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100:1–77, 1992.
20. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, 16:171–196, 1992.
21. U. Montanari and C. Talcott. Can actors and π-agents live together? In *Proc. HOOTS'97*, vol. 10 of *Elect. Notes in Th. Comput. Sci.*, Elsevier Science, 1998.
22. D. Park. Concurrency and automata on infinite sequences. In *Proc. 5th G-I Conference*, vol. 104 of *Lect. Notes in Comput. Sci.*, pages 167–183, Springer, 1981.
23. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.
24. P. Sewell. From rewrite rules to bisimulation congruences. In *Proc. CONCUR'98*, vol. 1466 of *Lect. Notes in Comput. Sci.*, pages 269–284, Springer, 1998.

Fibred Models of Processes: Discrete, Continuous, and Hybrid Systems
(Extended Abstract)

Marcelo P. Fiore

COGS, University of Sussex, Falmer, Brighton BN1 9QH, UK

Abstract. We present the rudiments of a unifying theory of general processes encompassing discrete, continuous, and hybrid systems. The main focus is on the study of process behaviour, but constructions on processes are also considered in some detail. In particular, we show that processes admit an abstract, conceptual treatment of bisimilarity via the notion of open map (as advocated by Winskel *et al.*). Furthermore, we present a tool-kit of categorical constructions on processes that can be regarded as the basis of a process description language. Within the general theory, typical operations of process calculi on discrete and hybrid systems are discussed.

Introduction

We advocate a fibred view of general processes; in which, roughly, a process

$$\begin{array}{c}\mathbb{X}\\ \downarrow\\ \mathbb{C}\end{array}$$

consists of a *state space* $\mathbb{X}$ (of states and their evolutions) varying according to a *control* $\mathbb{C}$ (of observations, time, *etc.*). By choosing the control $\mathbb{C}$ to consist of discrete observations or of continuous time we respectively obtain models of discrete and continuous processes (Section 1); hybrid systems arise from mixed discrete-continuous controls (Section 4). Further, we consider a *simulation*

$$\begin{array}{ccc}\mathbb{X} & & \mathbb{X}'\\ \downarrow & \longrightarrow & \downarrow\\ \mathbb{C} & & \mathbb{C}\end{array}$$

between processes to be a map $\mathbb{X} \longrightarrow \mathbb{X}'$ between the corresponding state spaces that respects the variation.

We will see that the category $\mathcal{U}$ of processes and simulations comes equipped with a canonical choice of *basic* processes (or paths) $\mathcal{P}$ from which arbitrary processes can be constructed by appropriately glueing basic ones. This situation yields a *model for concurrency*

$$\mathcal{P} \longrightarrow \mathcal{U}$$

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 457–473, 2000.
© Springer-Verlag Berlin Heidelberg 2000

in the sense of Winskel *et al.* [JNW96,CW97], and is important for one can study process behaviour via the notion of open map (Section 2). Moreover, we show that the category of processes $\mathcal{U}$ supports categorical constructions modelling typical operations of process calculi (Section 3).

We hope that this work will contribute towards the understanding of (and reasoning about) general processes and thereafter towards the development of a theory to aid the design of languages (and logics) for them.

1 Fibred Models of Discrete and Continuous Systems

We consider discrete and continuous systems from a fibred viewpoint. These systems are examples of *linearly-controlled* processes (or *interleaving models*, in the jargon of concurrency theory) in the sense of [BF99].

1.1 Discrete Systems

We study the familiar model of concurrent computation given by transition graphs (or automata) as an instance of the general theory to be developed in Section 2.

Transition graphs. For a set A, the category $\mathbf{TG}_A$ of *A-labelled transition graphs (or automata)* has objects given by graphs $\mathsf{dom}, \mathsf{cod} : E \longrightarrow N$ equipped with a labelling function $\mathsf{lab} : E \longrightarrow A$, and morphisms given by pairs of functions between nodes and edges that preserve the domain, codomain and labelling of edges. *Transition systems* are extensional transition graphs, for which the function $E \longrightarrow N \times A \times N : e \longmapsto (\mathsf{dom}(e), \mathsf{lab}(e), \mathsf{cod}(e))$ is injective.

Transition categories. Let $\mathbf{M}(A) = (A^*, \varepsilon, \cdot)$ be the free monoid on a set A. An *A-labelled transition category* is a functor $\ell\downarrow^{\mathbb{G}}_{\mathbf{M}(A)}$ such that, for every $e \in \mathbb{G}$ and $\alpha_0, \alpha_1 \in \mathbf{M}(A)$, if $\ell(e) = \alpha_0 \cdot \alpha_1$ in $\mathbf{M}(A)$ then there exists a unique factorisation $e = e_0 \cdot e_1$ in $\mathbb{G}$ for which $\ell(e_0) = \alpha_0$ and $\ell(e_1) = \alpha_1$. The category $\mathbf{TC}_A$ has A-labelled transition categories as objects and morphisms $\ell\downarrow^{\mathbb{G}}_{\mathbf{M}(A)} \longrightarrow \ell'\downarrow^{\mathbb{G}'}_{\mathbf{M}(A)}$ given by functors $\mathbb{G} \xrightarrow{h} \mathbb{G}'$ that preserve the labelling; that is, such that $\ell = \ell' h$.

Note that there is a bijective correspondence between transition graphs and transition categories as follows. A transition graph $G = (\ N \underset{\mathsf{cod}}{\overset{\mathsf{dom}}{\leftarrow\!\!\leftarrow}} E \xrightarrow{\mathsf{lab}} A\ )$ corresponds to the transition category $\ell\downarrow^{\mathcal{F}G}_{\mathbf{M}(A)}$, where $\mathcal{F}G$ is the free category on the graph G (with set of objects given by the nodes of G and morphisms given

by paths in G) and where $\ell(e_0 \cdots e_n) \stackrel{\text{def}}{=} \mathsf{lab}(e_0) \cdots \mathsf{lab}(e_n)$. Conversely, a transition category $\ell\!\downarrow^{\mathbb{G}}_{\mathbf{M}(A)}$ corresponds to the transition graph $|\,\mathbb{G}\,| \underset{\mathsf{cod}}{\overset{\mathsf{dom}}{\leftleftarrows}} E \xrightarrow{\ell} A$ where $E \stackrel{\text{def}}{=} \{\, e \in \mathbb{G} \mid \ell(e) \in A \,\}$.

Proposition 1.1. *For any set A, the categories $\mathbf{TG}_A$ and $\mathbf{TC}_A$ are equivalent.*

Thus, transition graphs and transition categories provide the same model of parallel computation. However, the fibred representation provided by transition categories embodies the dynamics of the systems. Indeed, as we show below, $\mathbf{TC}_A$ comes equipped with canonical categories of *computation paths* which induce behavioural notions of equivalence.

Discrete path categories. The category $\mathbf{M}(A)_{\lessapprox}$ has A^* as set of objects and morphisms $\alpha \longrightarrow \beta$ given by pairs (φ, γ) in A^* such that $\varphi \cdot \alpha \cdot \gamma = \beta$. One thinks of a morphism $\alpha \longrightarrow \beta$ as a pre- and post- extension of α yielding β. The category $\mathbf{M}(A)_{>}$ is the subcategory of $\mathbf{M}(A)_{\lessapprox}$ consisting of post-extensions; *viz.*, morphisms of the form (ε, γ), where ε denotes the empty string. Note that $\mathbf{M}(A)_{>}$ is the poset $\mathbf{P}(A) = (A^*, \leq)$ of sequences of elements of A ordered by prefix. We have faithful functors

$$\mathbf{P}(A) \longrightarrow \mathbf{M}(A)_{\lessapprox} \hookrightarrow \mathbf{TC}_A$$

where the embedding $\mathbf{M}(A)_{\lessapprox} \hookrightarrow \mathbf{TC}_A$ maps a sequence α to the transition category $\begin{array}{c} [\![\alpha]\!] \\ \downarrow \\ \mathbf{M}(A) \end{array} \quad \begin{array}{c} \alpha_0 \leq \alpha_0 \cdot \alpha_1 \\ \downarrow \\ \alpha_1 \end{array}$, where $[\![a_1 \cdots a_i \cdots a_n]\!]$ is the poset: $\varepsilon \leq a_1 \leq \ldots \leq (a_1 \cdots a_i) \leq \ldots \leq (a_1 \cdots a_i \cdots a_n)$.

Open-map bisimilarity. The functors $\mathbf{P}(A) \longrightarrow \mathbf{TC}_A$ and $\mathbf{M}(A)_{\lessapprox} \hookrightarrow \mathbf{TC}_A$ provide canonical choices w.r.t. which to consider the notion of open map [JNW96] and hence study process behaviour. Indeed, the notion of open map w.r.t. the inclusion $\mathbf{P}(A) \longrightarrow \mathbf{TC}_A$ corresponds to functional bisimulation (*cf.* [NC95]), whilst the notion of open map w.r.t. the embedding $\mathbf{M}(A)_{\lessapprox} \hookrightarrow \mathbf{TC}_A$ amounts to functional *back-and-forth* bisimulation on states (*cf.* [DNMV90]) —see the general theory developed in Section 2 '§ Bisimulation'. Moreover, the fibred setting allows a natural treatment of *weak bisimilarity*; this is shown in Subsection 3.1 '§ Saturation'.

1.2 Continuous Systems

We provide a treatment of continuous systems emphasising the similarities with the above view of discrete systems.

Durations. The notion of duration was introduced by Lawvere in connection to investigations of processes in continuum physics, see [Law86].

Let $\mathbf{R}$ be the additive monoid of non-negative real numbers. A *duration* is a functor $\begin{array}{c}\mathbb{X}\\ d\downarrow\\ \mathbf{R}\end{array}$ such that, for every $e \in \mathbb{X}$ and $t_0, t_1 \in \mathbf{R}$, if $d(e) = t_0 + t_1$ in $\mathbf{R}$ then there exists a unique factorisation $e = e_0 \cdot e_1$ in $\mathbb{X}$ for which $d(e_0) = t_0$ and $d(e_1) = t_1$. The category **Dur** has durations as objects and morphisms $\begin{array}{c}\mathbb{X}\\ d\downarrow\\ \mathbf{R}\end{array} \xrightarrow{h} \begin{array}{c}\mathbb{X}'\\ d'\downarrow\\ \mathbf{R}\end{array}$ given by functors $\mathbb{X} \xrightarrow{h} \mathbb{X}'$ such that $d = d'h$.

As remarked in [Law86], an important example of duration is given by solutions of differential equations. For the simplest such example, consider the differential equation

$$\frac{dx}{dt} = ax \tag{1}$$

whose initial value problem with initial condition

$$x(0) = s$$

has unique solution

$$\widetilde{s}(t) = s\, e^{at} \quad .$$

It can be regarded as the duration $\begin{array}{ccc}\mathbf{S} & & s_0 \xrightarrow{t} s_1\\ \downarrow & & \top\\ \mathbf{R} & & t\end{array}$, where $|\,\mathbf{S}\,| \stackrel{\text{def}}{=} \mathbb{R}$ and, for $t \in \mathbb{R}_{\geq 0}$, we have $s_0 \xrightarrow{t} s_1$ in $\mathbf{S}$ if and only if there exists a solution of (1) $\sigma : I \longrightarrow \mathbb{R}$ and $t_0 \leq t_1$ in I such that $\sigma(t_0) = s_0$, $\sigma(t_1) = s_1$, and $t_1 - t_0 = t$ (*i.e.*, if and only if $\widetilde{s_0}(t) = s_1$). (Compare this construction with the *flow* of the differential equation [HS74].)

Continuous path categories. The category $\mathbf{R}_{\gtrdot}$ has $\mathbb{R}_{\geq 0}$ (the set of non-negative real numbers) as the set of objects and morphisms $t \longrightarrow t'$ given by pairs (x, y) in $\mathbb{R}_{\geq 0}$ such that $x + t + y = t'$. A morphism $t \longrightarrow t'$ can be thought of as a way of placing an interval of length t within an interval of length t'. The category $\mathbf{R}_{>}$ is the subcategory of $\mathbf{R}_{\gtrdot}$ consisting of initial placements; *viz.*, morphisms of the form $(0, y)$. Note that $\mathbf{R}_{>}$ is the poset $\mathbf{T} = (\mathbb{R}_{\geq 0}, \leq)$. We have faithful functors

$$\mathbf{T} \longrightarrow \mathbf{R}_{\gtrdot} \hookrightarrow \mathbf{Dur}$$

where the embedding $\mathbf{R}_{\gtrdot} \hookrightarrow \mathbf{Dur}$ maps a non-negative real number t to the *difference* duration $\begin{array}{cc}[\![t]\!] & x \leq y\\ \downarrow & \downarrow\\ \mathbf{R} & y - x\end{array}$, where $[\![t]\!]$ is the *interval* poset $([0, t], \leq)$.

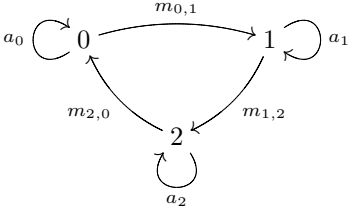
The concept of open map of durations, w.r.t. the path categories $\mathbf{T}$ and $\mathbf{R}_{\gtrdot}$, yields natural notions of bisimilarity; consider, for instance, the general results of Section 2 '§ Bisimulation' in the context of durations.

2 Fibred Models of Processes: General Theory

Processes. A *bundle* over a small category $\mathbb{C}$ is a functor $\begin{array}{c}\mathbb{X}\\ \downarrow\\ \mathbb{C}\end{array}$. The category $\mathbf{Cat}_{/\mathbb{C}}$ has bundles over $\mathbb{C}$ as objects and morphisms $\begin{array}{c}\mathbb{X}\\ f\downarrow\\ \mathbb{C}\end{array} \xrightarrow{\ h\ } \begin{array}{c}\mathbb{X}'\\ f'\downarrow\\ \mathbb{C}\end{array}$ given by functors $\mathbb{X} \xrightarrow{\ h\ } \mathbb{X}'$ such that $f = f'h$. The notion of process to be considered is given by certain bundles, known as ufl (*unique factorisation lifting*) functors (see , *e.g.*, [Law86,SS88,Str96]).

A *ufl functor* $\begin{array}{c}\mathbb{X}\\ f\downarrow\\ \mathbb{C}\end{array}$ (over a small category $\mathbb{C}$) is a functor with the property that, for every $e \in \mathbb{X}$ and $\alpha_0, \alpha_1 \in \mathbb{C}$, if $f(e) = \alpha_0 \cdot \alpha_1$ in $\mathbb{C}$ then there exists a unique factorisation $e = e_0 \cdot e_1$ in $\mathbb{X}$ for which $f(e_0) = \alpha_0$ and $f(e_1) = \alpha_1$. The category $\mathbf{Ufl}_{/\mathbb{C}}$ is the full subcategory of $\mathbf{Cat}_{/\mathbb{C}}$ with objects given by ufl functors.

The discrete and continuous systems of Section 1 provide examples of processes over monoids (*viz.*, $\mathbf{TC}_A = \mathbf{Ufl}_{/\mathbf{M}(A)}$ and $\mathbf{Dur} = \mathbf{Ufl}_{/\mathbf{R}}$) and as such force a *global* view of the state of a process. The extra generality of allowing processes over arbitrary categories may be used, for example, to provide a *distributed* view of the state. For instance, discrete processes operating in three modes (say $0, 1, 2$) and such that in mode i can either keep on computing in that mode or proceed to compute in mode $(i+1)$ mod 3, can be naturally modelled as ufl functors over the free category on the following graph of modes:



where, for $0 \leq i \leq 2$, the sets A_i are the observable actions in mode i and the transitions $m_{i,(i+1) \bmod 3}$ represent the allowable mode changes.

Path categories. The categories of processes $\mathbf{Ufl}_{/\mathbb{C}}$ come equipped with canonical categories of *paths* generalising the constructions of Section 1.

Mac Lane's *twisted arrow category* [ML71, page 223] $\mathbb{C}_{\twoheadrightarrow}$ has objects given by morphisms in $\mathbb{C}$ and morphisms $\alpha \longrightarrow \beta$ given by pairs of maps (φ, γ) in $\mathbb{C}$ such that $\varphi \cdot \alpha \cdot \gamma = \beta$; such a morphism can be regarded as a context, $\varphi \cdot (_) \cdot \gamma$, within which α can be placed to yield β. The category $\mathbb{C}_{>}$ is the subcategory of $\mathbb{C}_{\twoheadrightarrow}$ consisting of the morphisms of the form (id, γ). We have faithful functors

$$\mathbb{C}_{>} \xrightarrow{\ \mathrm{j}_{\mathbb{C}}\ } \mathbb{C}_{\twoheadrightarrow} \xhookrightarrow{\ \mathbf{u}_{\mathbb{C}}\ } \mathbf{Ufl}_{/\mathbb{C}} \tag{2}$$

where the embedding $\mathbb{C}_{\circledast} \hookrightarrow \mathbf{Ufl}_{/\mathbb{C}}$ maps:

- an object $(a_0 \xrightarrow{\alpha} a_1) \in |\,\mathbb{C}_{\circledast}\,|$ to the ufl functor $\mathbf{u}_\alpha \downarrow \begin{smallmatrix} [\![\alpha]\!] \\ \\ \mathbb{C} \end{smallmatrix}$ $\begin{smallmatrix} (\alpha_0, a, \alpha_1) \\ \top \\ \downarrow \\ a \end{smallmatrix}$, where the *interval category* $[\![\alpha]\!]$ has objects given by factorisations $(a_0 \xrightarrow{\alpha_0} a \xrightarrow{\alpha_1} a_1)$ of α in $\mathbb{C}$, and morphisms $\epsilon : (\alpha_0, a, \alpha_1) \longrightarrow (\alpha_0', a', \alpha_1')$ given by maps $\epsilon : a \longrightarrow a'$ in $\mathbb{C}$ such that $\alpha_0 \cdot \epsilon = \alpha_0'$ and $\alpha_1 = \epsilon \cdot \alpha_1'$; and
- a morphism $(\varphi, \gamma) : \alpha \longrightarrow \beta$ in $\mathbb{C}_{\circledast}$ to the functor $[\![\varphi, \gamma]\!] : [\![\alpha]\!] \longrightarrow [\![\beta]\!] : (\alpha_0, \alpha_1) \longmapsto (\varphi \cdot \alpha_0, \alpha_1 \cdot \gamma)$.

Note the (natural) bijective correspondence

$$\mathbf{Ufl}_{/\mathbb{C}}\,(\mathbf{u}_\alpha, f) \;\cong\; f^{-1}(\alpha) \qquad (3)$$
$$h \longmapsto h\left((\mathrm{id}, \alpha) \xrightarrow{\alpha} (\alpha, \mathrm{id})\right)$$

intuitively stating that computation paths (or *runs*) of shape α of the process f (*viz.*, morphisms $\mathbf{u}_\alpha \downarrow \begin{smallmatrix} [\![\alpha]\!] \\ \\ \mathbb{C} \end{smallmatrix} \longrightarrow f \downarrow \begin{smallmatrix} \mathbb{X} \\ \\ \mathbb{C} \end{smallmatrix}$) correspond to evolutions in the state space taking place over α (*viz.*, morphisms $e \in \mathbb{X}$ such that $f(e) = \alpha$).

In $\mathbb{C}_{\circledast}$, the commutative diagrams

$$\begin{array}{ccccc} & & \mathrm{id}_b & & \\ & {\scriptstyle(\alpha,\mathrm{id}_b)}\swarrow & & \searrow{\scriptstyle(\mathrm{id}_b,\beta)} & \\ \alpha & & & & \beta \\ & {\scriptstyle(\mathrm{id}_a,\beta)}\searrow & & \swarrow{\scriptstyle(\alpha,\mathrm{id}_c)} & \\ & & \alpha \cdot \beta & & \end{array} \qquad (a \xrightarrow{\alpha} b \xrightarrow{\beta} c \text{ in } \mathbb{C}) \qquad (4)$$

are pushouts, and the embedding $\mathbf{u}_\mathbb{C}$ exhibits $\mathbf{Ufl}_{/\mathbb{C}}$ as the free cocompletion of $\mathbb{C}_{\circledast}$ respecting these pushouts. More precisely, we have the following universal characterisation.

Theorem 2.1. *(cf. [BF99, § 2]) The category* $\mathbf{Ufl}_{/\mathbb{C}}$ *is cocomplete and the embedding* $\mathbf{u}_\mathbb{C} : \mathbb{C}_{\circledast} \hookrightarrow \mathbf{Ufl}_{/\mathbb{C}}$ *preserves the pushouts (4). Moreover, for every cocomplete category* $\mathcal{C}$ *and functor* $F : \mathbb{C}_{\circledast} \longrightarrow \mathcal{C}$ *preserving the pushouts (4), we have that*

1. *there exists a cocontinuous functor* $F' : \mathbf{Ufl}_{/\mathbb{C}} \longrightarrow \mathcal{C}$ *such that* $F'\mathbf{u}_\mathbb{C} \mathrel{\dot\cong} F$, *and*
2. *for every natural transformation* $\varphi : F'\mathbf{u}_\mathbb{C} \mathrel{\dot\longrightarrow} G'\mathbf{u}_\mathbb{C}$, *where* F' *and* G' *are cocontinuous functors* $\mathbf{Ufl}_{/\mathbb{C}} \longrightarrow \mathcal{C}$, *there exists a unique natural transformation* $\varphi' : F' \mathrel{\dot\longrightarrow} G'$ *such that* $\varphi'\mathbf{u}_\mathbb{C} = \varphi$.

Condition (1) asserts that every functor $\mathbb{C}_{\circledast} \longrightarrow \mathcal{C}$ preserving the pushouts (4) has (up to isomorphism) a cocontinuous extension $\mathbf{Ufl}_{/\mathbb{C}} \longrightarrow \mathcal{C}$; whilst condition (2) guarantees that such extensions are unique up to canonical isomorphism.

Theorem 2.1 is important for it shows that the processes in $\mathbf{Ufl}_{/\mathbb{C}}$ consist of *properly glued* copies of the basic processes, $\mathbf{u}_\alpha \downarrow_{\mathbb{C}}^{[\![\alpha]\!]}$, arising from computation paths. Formally, we have that, for every $f \downarrow_{\mathbb{C}}^{\mathbb{X}} \in |\, \mathbf{Ufl}_{/\mathbb{C}} \,|$,

$$f \cong \mathrm{colim}\left(\mathbb{X}_{\not\approx} \xrightarrow{f_{\not\approx}} \mathbb{C}_{\not\approx} \hookrightarrow^{\mathbf{u}_{\mathbb{C}}} \mathbf{Ufl}_{/\mathbb{C}} \right)$$

where $f_{\not\approx} : e \longmapsto f(e)$.

Pointed processes. For $u \downarrow_{\mathbb{C}}^{\mathbb{U}} \in |\, \mathbf{Ufl}_{/\mathbb{C}} \,|$, the category $u/\mathbf{Ufl}_{/\mathbb{C}}$ has: objects given by triples $(x, \mathbb{X}, f)$, where $x : \mathbb{U} \longrightarrow \mathbb{X}$ is a functor and $f \downarrow_{\mathbb{C}}^{\mathbb{X}}$ is a ufl functor, such that $fx = u$; and morphisms $(x, \mathbb{X}, f) \xrightarrow{h} (x', \mathbb{X}', f')$ given by functors $\mathbb{X} \xrightarrow{h} \mathbb{X}'$ such that $f = f'h$ and $hx = x'$.

For instance, $\downarrow_{\mathbb{C}}^{\mathbf{0}} / \mathbf{Ufl}_{/\mathbb{C}} \cong \mathbf{Ufl}_{/\mathbb{C}}$. Further, note that for $\alpha \in \mathbb{C}$, by the correspondence (3), $\mathbf{u}_\alpha/\mathbf{Ufl}_{/\mathbb{C}}$ is (isomorphic to) the category with objects ($e \in \mathbb{X}$, $f \downarrow_{\mathbb{C}}^{\mathbb{X}} \in |\, \mathbf{Ufl}_{/\mathbb{C}} \,|$) such that $f(e) = \alpha$, and with morphisms $(e, f) \xrightarrow{h} (e', f')$ given by maps $f \xrightarrow{h} f'$ in $\mathbf{Ufl}_{/\mathbb{C}}$ such that $h(e) = e'$. In particular, $\mathbf{u}_\varepsilon/\mathbf{Ufl}_{/\mathbf{M}(A)}$ is the category of transition categories with a designated object (typically modelling an initial state).

We remark that the situation (2) induces the following one:

$$\alpha/\mathbb{C}_{>} \longrightarrow \alpha/\mathbb{C}_{\not\approx} \hookrightarrow \mathbf{u}_\alpha/\mathbf{Ufl}_{/\mathbb{C}} \qquad (\alpha \in \mathbb{C})$$

between pointed path categories and pointed processes.

Linearly-controlled processes. We turn attention to *interleaving models*. These consist of processes varying over a category whose basic processes (or paths) are linear. Formally, we say that a category $\mathbb{C}$ is *path-linearisable* if, for every morphism $\alpha \in \mathbb{C}$, the interval category $[\![\alpha]\!]$ is a linear preorder, and we refer to ufl functors over path-linearisable categories as *linearly-controlled* process (see [BF99, § 4]).

Theorem 2.2. *([BF99, § 4]) A category $\mathbb{C}$ is path-linearisable if and only if the functor $[\![_]\!] : \mathbb{C}_{\not\approx} \longrightarrow \mathbf{Cat} : \alpha \longmapsto [\![\alpha]\!]$ preserves the pushouts (4).*

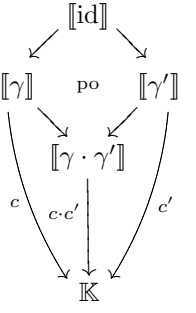
(In the setting of [BF99], this theorem corresponds to the equivalence between the properties (CFI) and (IG).)

Examples of path-linearisable categories are the monoids $\mathbf{M}(A)$ (for any set A) and $\mathbf{R}$, the posets $\mathbf{P}(A)$ (for any set A) and $\mathbf{T}$, and free categories on graphs. Moreover, path-linearisable categories are close under the sum and tensor (see '§ Tensor' in Section 3) of categories, but not under product.

There is a universal way in which to transform a bundle over a path-linearisable category into a linearly-controlled process. The construction is given by the proposition below.

Proposition 2.1. *([BN98]) For a path-linearisable category $\mathbb{C}$, the embedding $\mathbf{Ufl}_{/\mathbb{C}} \hookrightarrow \mathbf{Cat}_{/\mathbb{C}}$ has a right adjoint $\mathrm{U} : \mathbf{Cat}_{/\mathbb{C}} \longrightarrow \mathbf{Ufl}_{/\mathbb{C}}$.*

One can explicitly define $\mathrm{U}\left(\begin{array}{c}\mathbb{K}\\ \downarrow\\ \mathbb{C}\end{array}\right)$ as the first-projection functor $\begin{array}{c}\overline{\mathbb{K}}\\ \downarrow\\ \mathbb{C}\end{array}$ where $\overline{\mathbb{K}}$ has: objects given by pairs (C,k) with $C \in |\,\mathbb{C}\,|$ and $k \in \mathbf{Cat}_{/\mathbb{C}}\left(\begin{array}{cc}\llbracket \mathrm{id}_C \rrbracket & \mathbb{K}\\ \mathbf{u}_{\mathrm{id}_C}\downarrow & \downarrow\\ \mathbb{C} & \mathbb{C}\end{array}\right)$; morphisms $(C,k) \longrightarrow (C',k')$ given by pairs (γ, c) with $\gamma : C \longrightarrow C'$ in $\mathbb{C}$ and $c \in \mathbf{Cat}_{/\mathbb{C}}\left(\begin{array}{cc}\llbracket \gamma \rrbracket & \mathbb{K}\\ \mathbf{u}_{\gamma}\downarrow & \downarrow\\ \mathbb{C} & \mathbb{C}\end{array}\right)$ such that $k = c \circ \llbracket \mathrm{id}_C, \gamma \rrbracket$ and $c \circ \llbracket \gamma, \mathrm{id}_{C'} \rrbracket = k'$; identities $\mathrm{id}_{(C,k)} = (\mathrm{id}_C, k)$; and composition $(\gamma, c) \cdot (\gamma', c') = (\gamma \cdot \gamma', c \cdot c')$ given by the following construction



.

It follows from the proposition that, for $\mathbb{C}$ path-linearisable, colimits in $\mathbf{Ufl}_{/\mathbb{C}}$ are computed as in $\mathbf{Cat}_{/\mathbb{C}}$. In particular, we have the following corollary.

Corollary 2.1. *For a linearly-controlled process $\begin{array}{c}\mathbb{X}\\ f\downarrow\\ \mathbb{C}\end{array}$,*

$$\mathbb{X} \cong \mathrm{colim}\left(\mathbb{X}_{\lightning} \xrightarrow{f_{\lightning}} \mathbb{C}_{\lightning} \xrightarrow{\llbracket - \rrbracket} \mathbf{Cat}\right) \quad .$$

Bisimulation. For bundles $f\downarrow^{\mathbb{X}}_{\mathbb{C}}$ and $f'\downarrow^{\mathbb{X}'}_{\mathbb{C}}$, a relation $R \subseteq |\,\mathbb{X}\,| \times |\,\mathbb{X}'\,|$ is said to be a *simulation* (*cf.* [Mil89]) if $x_0\ R\ x'_0$ implies that, for every $x_0 \xrightarrow{e} x$ in $\mathbb{X}$, there exists $x'_0 \xrightarrow{e'} x'$ in $\mathbb{X}'$ such that $f(e) = f'(e')$ and $x\ R\ x'$.

Proposition 2.2. *For a path-linearisable category* $\mathbb{C}$*, consider* $f\downarrow^{\mathbb{X}}_{\mathbb{C}} \xrightarrow{h} f'\downarrow^{\mathbb{X}'}_{\mathbb{C}}$ *in* $\mathbf{Ufl}_{/\mathbb{C}}$*. The functor* h *is open w.r.t. the inclusion* $\mathbb{C}_{>} \longrightarrow \mathbf{Ufl}_{/\mathbb{C}}$ *if and only if, for every* $x_0 \in |\,\mathbb{X}\,|$ *and every* $h(x_0) \xrightarrow{e'} x'$ *in* $\mathbb{X}'$*, there exists* $x_0 \xrightarrow{e} x$ *in* $\mathbb{X}$ *such that* $h(e) = e'$*.*

For processes $\downarrow^{\mathbb{X}}_{\mathbb{C}}$ and $\downarrow^{\mathbb{X}'}_{\mathbb{C}}$, we say that $x \in |\,\mathbb{X}\,|$ and $x' \in |\,\mathbb{X}'\,|$ such that $f(x) = f'(x')$ are *open-map bisimilar* if there exists a span of functors

$$\downarrow^{\mathbb{X}}_{\mathbb{C}} \xleftarrow{h} \downarrow^{\mathbb{W}}_{\mathbb{C}} \xrightarrow{h'} \downarrow^{\mathbb{X}'}_{\mathbb{C}}$$

which are open w.r.t. the inclusion $\mathbb{C}_{>} \longrightarrow \mathbf{Ufl}_{/\mathbb{C}}$ such that $h(w) = x$ and $h'(w) = x'$, for some $w \in |\,\mathbb{W}\,|$.

Corollary 2.2. *For linearly-controlled processes, the notions of bisimilarity and of open-map bisimilarity coincide.*

Relation to presheaf models. The category of processes $\mathbf{Ufl}_{/\mathbb{C}}$ is intimately related to the presheaf categories over the path categories $\mathbb{C}_{\not\approx}$ (see [BF99]) and $\mathbb{C}_{>}$.

We have faithful functors

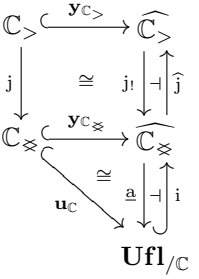
$$\mathbf{Ufl}_{/\mathbb{C}} \xhookrightarrow{\mathrm{i}} \widehat{\mathbb{C}_{\not\approx}} \xrightarrow{\hat{\mathrm{j}}} \widehat{\mathbb{C}_{>}} \quad , \tag{5}$$

where $\mathrm{i}(f) \stackrel{\text{def}}{=} \mathbf{Ufl}_{/\mathbb{C}}(\mathbf{u}_{(_)}, f)$ (*cf.* (3)) and $\hat{\mathrm{j}}(P) \stackrel{\text{def}}{=} P\,\mathrm{j}^{\mathrm{op}}$, that correspond to *unfolding* a process into the presheaf of its computations (or runs). This is clearly seen when $\mathbb{C} = \mathbf{M}\,(A)$, in which case $\widehat{\mathbb{C}_{>}} \cong \widehat{\mathbf{P}\,(A)}$ is (equivalent to) the category $\mathbf{SF}_A$ of A-labelled synchronisation forests (see [JNW96]) and the composite (5) amounts to the functor

$$\mathbf{TC}_A \longrightarrow \mathbf{SF}_A$$

that unfolds a transition graph into the forest of synchronisation trees rooted at each state (see [WN95,JNW96]).

In more detail, we have the following relationship between the fibred viewpoint and the presheaf models



where the functors $\mathrm{j}_!$ and $\underline{\mathrm{a}}$ are obtained as free cocontinous extensions. This diagram allows us to relate the various notions of open map (and hence of bisimilarity) as follows.

Proposition 2.3. *The functors* $\mathrm{j}_!$, $\underline{\mathrm{a}}$, i, *and* $\widehat{\mathrm{j}}$ *preserve open maps (w.r.t. the embeddings* $\mathbf{y}_{\mathbb{C}_>}$, $\mathbf{y}_{\mathbb{C}_{\not\geq}}$, *and* $\mathbf{u}_{\mathbb{C}}$*).*

Corollary 2.3. *A functor* h *in* $\mathbf{Ufl}_{/\mathbb{C}}$ *is* $\mathbf{u}_{\mathbb{C}}$*-open if and only if the map* $\mathrm{i}(h)$ *in* $\widehat{\mathbb{C}_{\not\geq}}$ *is* $\mathbf{y}_{\mathbb{C}_{\not\geq}}$*-open.*

3 Constructions on Processes

We present a tool-kit of categorical constructions on processes that can be regarded as the basis of an abstract process description language, *cf.* [WN95, § 2] and [CW97].

Composition. We consider two versions of operations obtained by post-composition.

1. For a ufl functor $g : \mathbb{A} \longrightarrow \mathbb{B}$, we define

$$\begin{array}{rcl} \Sigma_g : \mathbf{Ufl}_{/\mathbb{A}} & \longrightarrow & \mathbf{Ufl}_{/\mathbb{B}} \\ \mathbb{X}\downarrow f\downarrow \mathbb{A} & \longmapsto & \mathbb{X}\downarrow gf\downarrow \mathbb{B} \end{array}$$

2. For a functor $g : \mathbb{A} \longrightarrow \mathbb{B}$ where $\mathbb{B}$ is path-linearisable, we define

$$\begin{array}{rcl} \mathrm{U}_g : \mathbf{Ufl}_{/\mathbb{A}} & \longrightarrow & \mathbf{Ufl}_{/\mathbb{B}} \\ \mathbb{X}\downarrow f\downarrow \mathbb{A} & \longmapsto & \mathrm{U}\left(\mathbb{X}\downarrow gf\downarrow \mathbb{B}\right) \end{array}$$

We will also need the following operation given by pre-composition: for $h : u \longrightarrow v$ in $\mathbf{Ufl}_{/\mathbb{C}}$, we define

$$\begin{aligned}(_)|_h : v/\mathbf{Ufl}_{/\mathbb{C}} &\longrightarrow u/\mathbf{Ufl}_{/\mathbb{C}}\\ (y, \mathbb{Y}, g) &\longmapsto (yh, \mathbb{Y}, g)\end{aligned}$$

Pullback. For a functor $h : \mathbb{A} \longrightarrow \mathbb{B}$, we define the *pullback* (along h) functor

$$\begin{aligned}h^* : \mathbf{Ufl}_{/\mathbb{B}} &\longrightarrow \mathbf{Ufl}_{/\mathbb{A}}\\ \overset{\mathbb{X}}{f\downarrow}_{\mathbb{B}} &\longmapsto \overset{h^*\mathbb{X}}{h^*f\downarrow}_{\mathbb{A}}\end{aligned}$$

where h^*f is the first-projection functor from the subcategory $h^*\mathbb{X}$ of $\mathbb{A} \times \mathbb{X}$ with morphisms $(\alpha, e) : (a, x) \longrightarrow (a', x')$ such that $h(\alpha) = f(e)$.

Saturation. The operation of *saturation* (*cf.* [FCW99]) by a functor $h : \mathbb{A} \longrightarrow \mathbb{B}$, where $\mathbb{A}$ is path-linearisable, is given by the monad on $\mathbf{Ufl}_{/\mathbb{A}}$ induced by the following composite of adjoints:

$$\mathbf{Ufl}_{/\mathbb{A}} \underset{\longrightarrow}{\overset{U}{\underset{\top}{\longleftarrow}}} \mathbf{Cat}_{/\mathbb{A}} \underset{\Sigma_h}{\overset{h^*}{\underset{\top}{\longleftarrow}}} \mathbf{Cat}_{/\mathbb{B}} \ .$$

Product. The *product* functor is defined as follows

$$\begin{aligned}_ \times _ : \mathbf{Ufl}_{/\mathbb{A}} \times \mathbf{Ufl}_{/\mathbb{B}} &\longrightarrow \mathbf{Ufl}_{/\mathbb{A}\times\mathbb{B}}\\ \overset{\mathbb{X}}{f\downarrow}_{\mathbb{A}}, \overset{\mathbb{Y}}{g\downarrow}_{\mathbb{B}} &\longmapsto \overset{\mathbb{X}\times\mathbb{Y}}{f \times g\downarrow}_{\mathbb{A}\times\mathbb{B}}\end{aligned}$$

Sum. The *sum* functor is given by the construction

$$\begin{aligned}_ + _ : \mathbf{Ufl}_{/\mathbb{A}} \times \mathbf{Ufl}_{/\mathbb{B}} &\longrightarrow \mathbf{Ufl}_{/\mathbb{A}+\mathbb{B}}\\ \overset{\mathbb{X}}{f\downarrow}_{\mathbb{A}}, \overset{\mathbb{Y}}{g\downarrow}_{\mathbb{B}} &\longmapsto \overset{\mathbb{X}+\mathbb{Y}}{f + g\downarrow}_{\mathbb{A}+\mathbb{B}}\end{aligned}$$

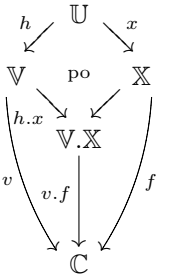
and induces the *fibred coproduct* functor $(_) \underset{\mathbb{C}}{+} (_)$ given by the composite

$$\mathbf{Ufl}_{/\mathbb{C}} \times \mathbf{Ufl}_{/\mathbb{C}} \xrightarrow{+} \mathbf{Ufl}_{/\mathbb{C}+\mathbb{C}} \xrightarrow{\Sigma_{[\mathrm{Id}_{\mathbb{C}}, \mathrm{Id}_{\mathbb{C}}]}} \mathbf{Ufl}_{/\mathbb{C}} \ .$$

Pushout. For $h : u \longrightarrow v$ in $\mathbf{Ufl}_{/\mathbb{C}}$ where $\mathbb{C}$ is path-linearisable, we have a *pushout* (along h) functor

$$\begin{aligned}h._ : u/\mathbf{Ufl}_{/\mathbb{C}} &\longrightarrow \quad v/\mathbf{Ufl}_{/\mathbb{C}}\\ (x, \mathbb{X}, f) &\longmapsto (h.x \ , \ \mathbb{V}.\mathbb{X} \ , \ v.f)\end{aligned}$$

defined by the diagram



.

Tensor. The *tensor* category (see, *e.g.*, [PR97, § 2] or [Str96, § 6]) $\mathbb{A} \otimes \mathbb{B}$ of the small categories $\mathbb{A}$ and $\mathbb{B}$ has objects given by pairs $a \otimes b$, with $a \in |\, \mathbb{A} \,|$ and $b \in |\, \mathbb{B} \,|$, and morphisms given by *interleaved (or shuffled) sequences* of non-identity composable maps in $\mathbb{A}$ and $\mathbb{B}$; *i.e.*, morphisms are formal sequences of non-identity morphisms of the form $a_1 \otimes b_1 \xrightarrow{\alpha_1 \oslash b_1} a_2 \otimes b_1 \xrightarrow{a_2 \oslash \beta_1} a_2 \otimes b_2 \longrightarrow \cdots$ or $a_1 \otimes b_1 \xrightarrow{a_1 \oslash \beta_1} a_1 \otimes b_2 \xrightarrow{\alpha_1 \oslash b_2} a_2 \otimes b_2 \longrightarrow \cdots$.

The tensor of categories has a universal characterisation as the following pushout square of functors

$$\begin{array}{ccc} |\, \mathbb{A} \,| \times |\, \mathbb{B} \,| & \xrightarrow{\mathrm{Id} \times \mathrm{J}_{\mathbb{B}}} & |\, \mathbb{A} \,| \times \mathbb{B} \\ \Big\downarrow{\scriptstyle \mathrm{J}_{\mathbb{A}} \times \mathrm{Id}} & \text{po} & \Big\downarrow{\scriptstyle - \oslash =} \\ \mathbb{A} \times |\, \mathbb{B} \,| & \xrightarrow[- \oslash =]{} & \mathbb{A} \otimes \mathbb{B} \end{array} \quad ,$$

from which one easily sees that the tensor category of two monoids is their coproduct in the category of monoids. Thus, in particular,

$$\mathbf{M}\,(A) \otimes \mathbf{M}\,(B) \cong \mathbf{M}\,(A + B) \quad .$$

Moreover, the construction induces a *tensor* functor between categories of processes:

$$\begin{array}{rcl} - \otimes = : \mathbf{Ufl}_{/\mathbb{A}} \times \mathbf{Ufl}_{/\mathbb{B}} & \longrightarrow & \mathbf{Ufl}_{/\mathbb{A} \otimes \mathbb{B}} \\ \overset{\mathbb{X}}{f}\Big\downarrow_{\mathbb{A}}\,,\ \overset{\mathbb{Y}}{g}\Big\downarrow_{\mathbb{B}} & \longmapsto & \overset{\mathbb{X} \otimes \mathbb{Y}}{f \otimes g}\Big\downarrow_{\mathbb{A} \otimes \mathbb{B}} \end{array}$$

3.1 Examples

We show that, for transition categories, the above categorical constructions yield typical operations of process calculi (*cf.* [WN95]). Hybrid systems are discussed in Section 4.

Relabelling. The operation of *relabelling* according to a function $\rho : A \longrightarrow B$ corresponds to the functor

$$_\{\rho\} \stackrel{\text{def}}{=} \Sigma_{\mathbf{M}(\rho)}(_) : \mathbf{Ufl}_{/\mathbf{M}(A)} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(B)}$$

where $\mathbf{M}(\rho)$ is the free homomorphic extension of the function mapping: $a \in A \longmapsto \rho(a)$.

Restriction. Let $A \subseteq B$, and write $\iota : A \longrightarrow B$ for the inclusion function. The *restriction* operation corresponds to the functor

$$_ \restriction \iota \stackrel{\text{def}}{=} \mathbf{M}(\iota)^* (_) : \mathbf{Ufl}_{/\mathbf{M}(B)} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A)} \ .$$

Saturation. The main example of a saturation monad is given by the construction that associates a transition graph with the transition graph obtained from saturating transitions by the hidden insertion of silent actions; *i.e.*, the construction introduced by Milner to define weak bisimulation (see [Mil89]).

In our setting (as in [FCW99], where an account of weak bisimilarity for presheaf models required, crucially, the adoption of a fibred view of processes), the saturation by silent actions arises as the saturation monad (of Section 3 '§ Saturation') by the homomorphism $\mathbf{M}(A + \{\tau\}) \longrightarrow \mathbf{M}(A)$ that *hides* the silent τ actions (*viz.*, the free homomorphic extension of the function mapping: $a \in A \longmapsto a$, $\tau \longmapsto \varepsilon$).

Generalising from the above discussion, we have the following general definition of weak bisimulation w.r.t. a hiding functor $h : \mathbb{A} \longrightarrow \mathbb{B}$: for processes $f \downarrow^{\mathbb{X}}_{\mathbb{A}}$ and $f' \downarrow^{\mathbb{X}'}_{\mathbb{A}}$, we say that $x \in |\, \mathbb{X} \,|$ and $x' \in |\, \mathbb{X}' \,|$ are *h-bisimilar* (*cf.* [FCW99]) whenever $\eta_f(x)$ and $\eta_{f'}(x')$ are open-map bisimilar, where η denotes the unit of the h-saturation monad.

Parallel composition. We discuss synchronous and asynchronous compositions of processes.

Synchrony. As observed in [WN95, § 2.2.4], versions of parallel composition can be obtained from the product operation by restriction and relabelling. Roughly, the *synchronous parallel composition* of processes can be specified as the combined operation

$$\Sigma_\rho(\sigma^*(_ \times _)) : \mathbf{Ufl}_{/\mathbb{A}} \times \mathbf{Ufl}_{/\mathbb{B}} \longrightarrow \mathbf{Ufl}_{/\mathbb{C}}$$

where $\sigma : \mathbb{S} \longrightarrow \mathbb{A} \times \mathbb{B}$ is a *syncronisation* functor and $\rho : \mathbb{S} \longrightarrow \mathbb{C}$ a *relabelling* ufl functor.

Synchronisation à la CCS is given by the functor

$$(\sigma^*(_ \times _))\,\{\rho\} : \mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\}+B)} \times \mathbf{Ufl}_{/\mathbf{M}(B+\{\tau\}+C)} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\}+C)}$$

where the synchronisation map

$$\mathbf{M}\,(A + \{(\tau,*)\} + B + \{(*,\tau)\} + C) \xrightarrow{\sigma} \mathbf{M}\,(A + \{\tau\} + B) \times \mathbf{M}\,(B + \{\tau\} + C)$$

is the free homomorphic extension of the function mapping: $a \in A \longmapsto (a,\varepsilon)$, $(\tau,*) \longmapsto (\tau,\varepsilon)$, $b \in B \longmapsto (b,b)$, $(*,\tau) \longmapsto (\varepsilon,\tau)$, $c \in C \longmapsto (\varepsilon,c)$; and where the relabelling function

$$A + \{(\tau,*)\} + B + \{(*,\tau)\} + C \xrightarrow{\rho} A + \{\tau\} + C$$

is the following mapping: $a \in A \longmapsto a$, $(\tau,*) \longmapsto \tau$, $b \in B \longmapsto \tau$, $(*,\tau) \longmapsto \tau$, $c \in C \longmapsto c$.

On the other hand, *synchronisation à la CSP*, corresponds to

$$(\sigma^*(_ \times _))\,\{\rho\} : \mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\})} \times \mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\})} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\})}$$

where the synchronisation map

$$\mathbf{M}\,(A + \{(\tau,*)\} + \{(*,\tau)\}) \xrightarrow{\sigma} \mathbf{M}\,(A + \{\tau\}) \times \mathbf{M}\,(A + \{\tau\})$$

is the free homomorphic extension of the function mapping: $a \in A \longmapsto (a,a)$, $(\tau,*) \longmapsto (\tau,\varepsilon)$, $(*,\tau) \longmapsto (\varepsilon,\tau)$; and where the relabelling function

$$A + \{(\tau,*)\} + \{(*,\tau)\} \xrightarrow{\rho} A + \{\tau\}$$

is the following mapping: $a \in A \longmapsto a$, $(\tau,*) \longmapsto \tau$, $(*,\tau) \longmapsto \tau$.

Asynchrony. The *asynchronous parallel composition* of processes can be described along the above lines as the functor

$$\sigma^* : \mathbf{Ufl}_{/\mathbf{M}(A)} \times \mathbf{Ufl}_{/\mathbf{M}(B)} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A+B)}$$

where the synchronisation map

$$\mathbf{M}\,(A + B) \xrightarrow{\sigma} \mathbf{M}\,(A) \times \mathbf{M}\,(B)$$

is the free homomorphic extension of the function mapping: $a \in A \longmapsto (a,\varepsilon)$, $b \in B \longmapsto (\varepsilon,b)$.

However, there is a more direct description; namely, as the tensor functor

$$\mathbf{Ufl}_{/\mathbf{M}(A)} \times \mathbf{Ufl}_{/\mathbf{M}(B)} \xrightarrow{\otimes} \mathbf{Ufl}_{/\mathbf{M}(A)\otimes\mathbf{M}(B)}\ .$$

Non-deterministic sum. For sets A and B, let $\iota_A : A \longrightarrow A \cup B$ and $\iota_B : B \longrightarrow A \cup B$ be the respective inclusion functions.

The *non-deterministic sum* of transition categories is given by the following construction:

$$\left(_ \underset{\mathbf{M}(A\cup B)}{+} _\right) \circ (_\,\{\iota_A\}\,,_\,\{\iota_B\}) : \mathbf{Ufl}_{/\mathbf{M}(A)} \times \mathbf{Ufl}_{/\mathbf{M}(B)} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A\cup B)}\ .$$

Prefix. The *prefix* operation on transition categories, for $a \in A$, arises as the composite

$$\left((_)|_{[\![\varepsilon,a]\!]}\right) \circ ([\![a,\varepsilon]\!]._) : \mathbf{u}_\varepsilon/\mathbf{Ufl}_{/\mathbf{M}(A)} \longrightarrow \mathbf{u}_\varepsilon/\mathbf{Ufl}_{/\mathbf{M}(A)}\ .$$

4 Fibred Models of Hybrid Systems

Hybrid systems are typically described as *mixed discrete-continuous* systems, and our development has been aimed at putting this slogan in mathematical form.

We have considered the categories $\mathbf{Ufl}_{/\mathbb{C}}$ as fibred models of processes, and observed, in Subsections 1.1 and 1.2, that for $\mathbb{C} = \mathbf{M}(A)$ the resulting processes correspond to discrete systems, whilst for $\mathbb{C} = \mathbf{R}$ they correspond to continuous ones. Models of hybrid systems arise from considering categories $\mathbb{C}$ where discrete actions and continuous evolutions are interleaved. The simplest example is the model

$$\mathbf{Ufl}_{/\mathbf{M}(A)\otimes\mathbf{R}}$$

consisting of processes $\begin{array}{c}\mathbb{X}\\ \downarrow\\ \mathbf{M}(A)\otimes\mathbf{R}\end{array}$ where the evolutions in the state space $\mathbb{X}$ factor uniquely as interleavings (or shuffles) of discrete and continuous evolutions. Examples of such processes arise from *hybrid automata* (*cf.* [Hen96, Definition 1.3]).

The models of discrete, continuous, and hybrid systems introduced relate as follows

$$\begin{array}{ccc} \mathbf{Ufl}_{/\mathbf{M}(A)} & & \mathbf{Ufl}_{/\mathbf{R}} \\ \circlearrowleft\ \Pi_1{}^* \ \top\ \Sigma_{\Pi_1} \ \circlearrowright & & \circlearrowleft\ \Pi_2{}^* \ \top\ \Sigma_{\Pi_2} \ \circlearrowright \\ & \mathbf{Ufl}_{/\mathbf{M}(A)\otimes\mathbf{R}} & \end{array} .$$

Thus the notions of open-map (and hence of bisimilarity) for hybrid systems extend that of discrete and continuous systems.

We remark that other *non-standard* models of hybrid systems can also be accomodated. For instance, hybrid systems with two modes of operation (discrete and continuous) where mode changes are explicit, can be seen as processes over the category generated by the diagram

$$a \circlearrowleft D \underset{d}{\overset{c}{\rightleftarrows}} C \circlearrowright t \quad (a \in A, t \in \mathbb{R}_{\geq 0})$$

under the condition that the composite $t \cdot t'$ be $t + t'$, for all $t, t' \in \mathbb{R}_{\geq 0}$.

4.1 Constructions on Hybrid Systems

We exemplify the use of the constructions of Section 3 on hybrid systems.

Parallel composition. The following operation corresponds to the *parallel composition* of hybrid systems as considered in [Hen96, § 1.4]:

$$\sigma^*(_ \times _) : \mathbf{Ufl}_{/\mathbf{M}(A+B)\otimes\mathbf{R}} \times \mathbf{Ufl}_{/\mathbf{M}(B+C)\otimes\mathbf{R}} \longrightarrow \mathbf{Ufl}_{/\mathbf{M}(A+B+C)\otimes\mathbf{R}}$$

where the synchronisation map

$$\mathbf{M}\,(A+B+C)\otimes\mathbf{R}\xrightarrow{\sigma}(\mathbf{M}\,(A+B)\otimes\mathbf{R})\times(\mathbf{M}\,(B+C)\otimes\mathbf{R})$$

is the unique homomorphism such that: $a\in A\longmapsto(a,\varepsilon)$, $b\in B\longmapsto(b,b)$, $c\in C\longmapsto(\varepsilon,c)$, $t\in\mathbb{R}_{>0}\longmapsto(t,t)$.

Time abstraction. We consider two versions of *time abstraction*. They both arise as the operation

$$\mathrm{U}_{\overline{\rho}}\circ\overline{h}^{*}:\mathbf{Ufl}_{/\mathbf{M}(A)\otimes\mathbf{R}}\longrightarrow\mathbf{Ufl}_{/\mathbf{M}(A+\{\tau\})}\ ,$$

where the diagram

$$\begin{array}{ccc} M & \xrightarrow{\ \overline{h}\ } & \mathbf{M}\,(A)\otimes\mathbf{R} \\ {\scriptstyle\overline{\rho}}\downarrow & & \downarrow{\scriptstyle\rho} \\ \mathbf{M}\,(A+\{\tau\}) & \xrightarrow[\ h\ ]{} & \mathbb{C} \end{array}$$

is a pullback square, for suitable choices of ρ and h.

1. The following version of time abstraction corresponds to the one considerd in [Hen96, Definition 1.3]. Take $\mathbb{C}=\mathbf{M}\,(A)\otimes\mathbf{E}$, where $\mathbf{E}$ is the monoid given by the following table

$$\begin{array}{|c|c|c|}\hline \cdot_{\mathbf{E}} & \mathrm{id} & \tau \\ \hline \mathrm{id} & \mathrm{id} & \tau \\ \hline \tau & \tau & \tau \\ \hline \end{array}\ ,$$

   together with $\rho:\mathbf{M}\,(A)\otimes\mathbf{R}\longrightarrow\mathbf{M}\,(A)\otimes\mathbf{E}$ the unique homomorphism mapping: $a\in A\longmapsto a$, $t\in\mathbb{R}_{>0}\longmapsto\tau$ and $h:\mathbf{M}\,(A+\{\tau\})\longrightarrow\mathbf{M}\,(A)\otimes\mathbf{E}$ the free homomorphic extension of the function mapping: $a\in A\longmapsto a$, $\tau\longmapsto\tau$.
2. A slightly simpler time-abstraction operation is obtained by taking $\mathbb{C}=\mathbf{M}\,(A)$ together with $\rho:\mathbf{M}\,(A)\otimes\mathbf{R}\longrightarrow\mathbf{M}\,(A)$ the unique homomorphism mapping: $a\in A\longmapsto a$, $t\in\mathbb{R}_{\geq 0}\longmapsto\varepsilon$ and $h:\mathbf{M}\,(A+\{\tau\})\longrightarrow\mathbf{M}\,(A)$ the free homomorphic extension of the function mapping: $a\in A\longmapsto a$, $\tau\longmapsto\varepsilon$.

   In this case, for a hybrid system $\begin{array}{c}\mathbb{X}\\ {\scriptstyle f}\downarrow\\ \mathbf{M}\,(A)\otimes\mathbf{R}\end{array}$, the transition category $\mathrm{U}_{\overline{\rho}}\left(\overline{h}^{*}f\right)$ obtained by time abstraction corresponds to the transition graph with set of nodes $|\,\mathbb{X}\,|$, set of edges $\{\,(a,e)\in A\times\mathbb{X}\mid f(e)=a\vee f(e)=t\cdot a\vee f(e)=a\cdot t'\vee f(e)=t\cdot a\cdot t'$, for some $t,t'\in\mathbb{R}_{>0}\,\}\cup\{\,(\tau,e)\in\{\tau\}\times\mathbb{X}\mid f(e)=\varepsilon\vee f(e)=t$, for some $t\in\mathbb{R}_{>0}\,\}$, with domains and codomains inherited from $\mathbb{X}$ and with labelling function given by first projection.

References

[BF99] M. Bunge and M.P. Fiore. Unique factorisation lifting functors and categories of linearly-controlled processes. To appear in the special issue of *Mathematical Structures in Computer Science* in honour of Joachim Lambek's 75^{th} birthday, 1999.

[BN98] M. Bunge and S. Niefield. Exponentiability and single universes. To appear in the *Journal of Pure and Applied Algebra*, 1998.

[CW97] G.L. Cattani and G. Winskel. Presheaf models for concurrency. In *Proceedings of CSL'96*, volume 1258 of *Lecture Notes in Computer Science*, pages 58–75. Springer-Verlag, 1997.

[DNMV90] R. De Nicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations. In *Proceedings of the CONCUR'90 Conf.*, volume 458 of *Lecture Notes in Computer Science*, pages 152–165. Springer-Verlag, 1990.

[FCW99] M.P. Fiore, G.L. Cattani, and G. Winskel. Weak bisimulation and open maps. In *14^{th} LICS Conf.*, pages 67–76. IEEE, Computer Society Press, 1999.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *11^{th} LICS Conf.*, pages 278–292. IEEE, Computer Society Press, 1996.

[HS74] M.W. Hirsch and S. Smale. *Differential equations, dynamical systems, and linear algebra*, volume 60 of *Pure and applied mathematics*. Academic Press, 1974.

[JNW96] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.

[Law86] F.W. Lawvere. Introduction. In F.W. Lawvere and S.H. Schanuel, editors, *Categories in continuum physics*, volume 1174 of *Lecture Notes in Mathematics*, pages 1–16. Springer-Verlag, 1986.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[ML71] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[NC95] M. Nielsen and A. Cheng. Observe behaviour categorically. In *Proceedings of FST&TCS '95*, volume 1026 of *Lecture Notes in Computer Science*, pages 263–278. Springer-Verlag, 1995.

[PR97] J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997.

[SS88] S. Schanuel and R. Street. The length of a pasting diagram. Unpublished, June 1988.

[Str96] R. Street. Categorical structures. In *Handbook of algebra*, volume 1, pages 529–577. North-Holland, 1996.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of logic in computer science, Vol. 4*, Oxford Sci. Publ., pages 1–148. Oxford Univ. Press, 1995.

On the Complexity of Bisimulation Problems for Pushdown Automata

Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France. E-mail: mayr@liafa.jussieu.fr
Phone: +33 1 44 27 28 40, Fax: +33 1 44 27 68 49

Abstract. All bisimulation problems for pushdown automata are at least *PSPACE*-hard. In particular, we show that (1) Weak bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, even for a small fixed finite automaton, (2) Strong bisimilarity of pushdown automata and finite automata is *PSPACE*-hard, but polynomial for every fixed finite automaton, (3) Regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity is *PSPACE*-hard.

Keywords: Pushdown automata, bisimulation, verification, complexity

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [21]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [22] for a survey). While many algorithms for bisimulation problems have a very high complexity, only few lower bounds are known. Jančar [12,13] showed that strong bisimilarity of two Petri nets [25] and weak bisimilarity of a Petri net and a finite automaton is undecidable. Stříbrná [28] showed that weak bisimilarity for Basic Parallel Processes (BPP) is $\mathcal{NP}$-hard and weak bisimilarity for context-free processes (BPA) is *PSPACE*-hard. (BPA are a proper subclass of pushdown automata.) However, it is still an open question whether these two problems are decidable. So far, the only known lower bound for a decidable bisimulation problem was an *EXPSPACE*-lower bound for strong bisimilarity of Petri nets and finite automata [15], that follows from the hardness of the Petri net reachability problem [18].

For bisimulation problems where one compares an infinite-state system with a finite-state one, much more is known about the decidability and complexity than in the general case of two infinite-state systems [14]. Also the complexity can be much lower. In particular, weak (and strong) bisimilarity of a BPA-process and a finite automaton is decidable in polynomial time [17], while weak bisimilarity of two BPA-processes is *PSPACE*-hard [28].

However, this surprising result does not carry over to general pushdown automata. We show that strong and weak bisimilarity of a pushdown automaton

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 474–488, 2000.
© Springer-Verlag Berlin Heidelberg 2000

and a finite automaton is *PSPACE*-hard. (These problems were already known to be in *EXPTIME* [14].) For weak bisimilarity this hardness result holds even for a small fixed finite automaton, while the same problem for strong bisimilarity is polynomial in the size of the pushdown automaton for every fixed finite automaton. These results also yield a *PSPACE* lower bound for strong bisimilarity of two pushdown automata, a problem that has recently been shown to be decidable by Sénizergues [27] (the proof in [27] uses a combination of two semidecision procedures and does not yield any complexity measure).

The problem of bisimilarity is also related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata [26]. See Section 5 for details.

Furthermore, we prove a *PSPACE* lower bound for the problem of regularity (finiteness) of pushdown automata w.r.t. weak and strong bisimilarity.

Thus no bisimulation problem for pushdown automata is polynomial (unless *PSPACE* is $\mathcal{P}$). This shows that there is a great difference between pushdown automata and BPA, although they describe exactly the same class of languages (Chomsky-2).

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E ::= \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '.' is a sequential composition and '$\|$' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '$\|$' are associative, '$\|$' is commutative, and 'ϵ' is a unit for '.' and '$\|$'.

A *process rewrite system* (PRS) [20] is specified by a finite set Δ of *rules* which have the form $E \stackrel{a}{\to} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \stackrel{a}{\to} F) \in \Delta}{E \stackrel{a}{\to} F} \qquad \frac{E \stackrel{a}{\to} E'}{E.F \stackrel{a}{\to} E'.F} \qquad \frac{E \stackrel{a}{\to} E'}{E \| F \stackrel{a}{\to} E' \| F}$$

We extend the notation $E \stackrel{a}{\to} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \stackrel{w}{\to} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. We also use '1' to denote the set of process constants.

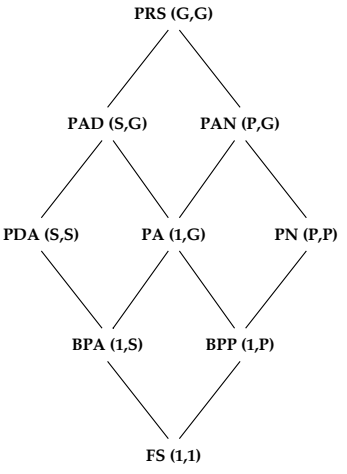


Fig. 1. A hierarchy of PRS

The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA (Basic Process Algebra, also called context-free processes), BPP (Basic Parallel Processes), and PA-processes are well-known [1], PDA correspond to pushdown automata (as proved by Caucal in [6]), PN correspond to Petri nets, PRS stands for 'Process Rewrite Systems', PAD and PAN are artificial names made by combining existing ones (PAD = PA+PDA, PAN = PA+PN).

We consider the semantical equivalences *weak bisimilarity* and *strong bisimilarity* [21] over transition systems generated by PRS. In what follows we consider process expressions over $Const(\Delta)$ where Δ is some fixed process rewrite system.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation *'$\stackrel{a}{\Rightarrow}$' is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over process expressions is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them.*

Bisimulation equivalence can also be described by *bisimulation games* between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

Note that context-free processes (BPA) correspond to the subclass of pushdown automata (PDA) where the finite control has size 1. Although BPA and PDA describe the same class of languages (Chomsky-2), BPA is strictly less expressive w.r.t. bisimulation.

3 Hardness of Weak Bisimulation Problems

In this section we show lower bounds for problems about weak bisimulation. We consider the following two problems:

Weak bisimilarity of pushdown automata and finite automata

Instance: A pushdown automaton P and a finite automaton F.
Question: $P \approx F$?

Weak Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \approx F$?

We show that both these problems are *PSPACE*-hard. The proof is done by a reduction from the *PSPACE*-complete problem if a single tape, linearly space-bounded, nondeterministic Turing-machine M accepts a given input w. There is a constant k s.t. if M accepts an input w then it has an accepting computation that uses only $k \cdot |w|$ space. For any such M and w we construct a pushdown automaton P s.t.

- If M accepts w then P is not weakly bisimilar to any finite automaton.
- If M doesn't accept w then P is weakly bisimilar to the finite automaton F of Figure 2.

The construction of P is as follows: Let $n := k \cdot |w| + 1$ and Σ be the set of tape symbols of M. Configurations of M are encoded as sequences of n symbols of the form $v_1 q v_2$ where $v_1, v_2 \in \Sigma^*$ are sequences of tape symbols of M and q is a state of the finite control of M. The sequence v_1 are the symbols to the left of the head and v_2 are the symbols under the head and to the right of it. (v_1 can be empty, but v_2 can't.) Let p_0 be the initial control-state of P and let the stack be initially empty. Initially, P is in the phase 'guess' where it guesses an arbitrarily long sequence $c_1 \# c_2 \# \ldots \# c_m$ of configurations of M (each of these c_i has length n) and stores them on the stack. The pushdown automaton can guess a sequence of length n by n times guessing a symbol and storing it on the stack. The number of symbols guessed (from 1 to n) is counted in the finite-control of the pushdown automaton. The number m is not counted in the finite-control, since it can be arbitrarily large. The configuration c_m at the bottom of the stack must be accepting (i.e., the state q in c_m must be accepting) and the configuration c_1 at the top must be the initial configuration with the input w and the initial control-state of M. All this is done with silent τ-actions. At the end of this phase P is in the control state p. Then there are two possible transitions: (1) $p \xrightarrow{\tau} p_0 A$ where the special symbol $A \notin \Sigma$ is written on the stack and the guessing phase starts again. (2) $p \xrightarrow{\tau} p_{verify}$ where the pushdown automaton enters the new phase 'verify'.

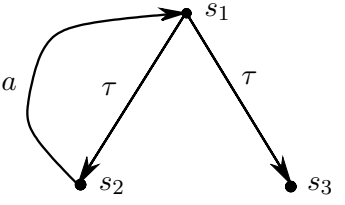


Fig. 2. The finite automaton F with initial state s_1.

In the phase 'verify' the pushdown automaton P pops symbols from the stack (by action τ). At any time in this phase it can (but need not) enter the special phase 'check'. For a 'check' it reads three symbols from the stack. These symbols are part of some configuration c_i. Then it pops $n-2$ symbols and then reads the three symbols at the same position in the next configuration c_{i+1} (unless the bottom of the stack is reached already). In a correct computation step from c_i to c_{i+1} the second triple of symbols depends on the first and on the definition of M. If these symbols in the second triple are as they should be in a correct computation step of M from c_i to c_{i+1} then the 'check' is successful and it goes back into the phase 'verify'. Otherwise the 'check' has failed and P is in the control-state *fail*. Here there are two possible transitions: (1) $\mathit{fail} \xrightarrow{\tau} p_2$. In the control-state p_2 the stack is ignored and the pushdown automaton from then on behaves just like the state s_2 in the finite automaton F of Figure 2. (2) $\mathit{fail} \xrightarrow{\tau} p_3$. In the control-state p_3 again the stack is ignored and from then on the pushdown automaton behaves just like the state s_3 in the finite automaton F of Figure 2. The intuition is that if the sequence of configurations represents a correct computation of M then no 'check' can fail, i.e., the control-state *fail* cannot be reached. However, if the sequence isn't a correct computation then there must be at least one error somewhere and thus the control-state *fail* can be reached by doing the 'check' at the right place.

So far, all actions have been silent τ-actions. The only case where a visible action can occur is the following: The pushdown automaton P is in phase 'verify' or 'check' (but not in state *fail*) and reads the special symbol A from the stack. Then it does the visible action 'a' and goes to the control-state p_{verify}. If P reaches the bottom of the stack while being in phase 'verify' or 'check' then it is in a deadlock.

Lemma 2. *If M accepts the input w then P is not weakly bisimilar to any finite automaton.*

Proof. We assume the contrary and derive a contradiction. Assume that there is finite automaton F' with k states s.t. $P \approx F'$. Since M accepts w, there exists an accepting computation sequence $c = c_1 \# c_2 \# \ldots \# c_m$ where all c_i are configurations of M, c_1 is the initial configuration of M with input w, c_m is accepting and for all $i \in \{1, \ldots, m-1\}$ $c_i \to c_{i+1}$ is a correct computation step of M.

P can (by a sequence of τ-steps) reach the configuration $\alpha := p_{verify}\,(cA)^{k+1}c$. Since c is an accepting computation sequence of M, none of the checks can fail. Thus α can only do the following sequence of actions: $\tau^{mn+m-1}(a\tau^{mn+m-1})^{k+1}$.

We assumed that $P \approx F'$. Thus there must be some state f of F' s.t. $\alpha \approx f$. Since F' has only k states, it follows from the Pumping Lemma for regular languages that $\alpha \not\approx f$ and we have a contradiction. □

Lemma 3. *Let F be the finite automaton from Figure 2. If M doesn't accept the input w then $P \approx F$.*

Proof. Since there is no accepting computation of M on w, any reachable configuration of P belongs to one of the following three sets.

1. Let C_1 be the set of configurations of P where either P is in phase 'guess' or P is in phase 'verify' or 'check' s.t. a check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* can be reached with only τ-actions.
2. Let C_2 be the set of configurations of P where either the finite control of P is in state p_2 or P is in phase 'verify' or 'check', there is at least one symbol A on the stack and no check can fail before the next symbol A is popped from the stack, i.e. the control-state *fail* cannot be reached with only τ-actions, but possibly after another 'a' action.
3. Let C_3 be the set of configurations of P where either the finite control of P is in state p_3 or P is in phase 'verify' or 'check', there is no symbol A on the stack and no check can fail, i.e. the control-state *fail* cannot be reached.

The following relation is a weak bisimulation:

$$\{(\alpha_1, s_1) \mid \alpha_1 \in C_1\} \cup \{(\alpha_2, s_2) \mid \alpha_2 \in C_2\} \cup \{(\alpha_3, s_3) \mid \alpha_3 \in C_3\}$$

We consider all possible attacks.

1. Note that no $\alpha_1 \in C_1$ can do action 'a'.
   - If the attacker makes a move from a configuration in C_1 with control-state *fail* to p_2/p_3 then the defender responds by a move $s_1 \stackrel{\tau}{\to} s_1/s_2$. These are weakly bisimilar to p_2/p_3 by definition. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\to} \alpha_1'$ with $\alpha_1, \alpha_1' \in C_1$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\to} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_2$ (this is only possible if there is at least one symbol A on the stack) then the defender responds by making a move $s_1 \stackrel{\tau}{\to} s_2$. If the attacker makes a move $\alpha_1 \stackrel{\tau}{\to} \alpha_1'$ with $\alpha_1 \in C_1$ and $\alpha_2 \in C_3$ (this is only possible if there is no symbol A on the stack) then the defender responds by making a move $s_1 \stackrel{\tau}{\to} s_3$.
   - If the attacker makes a move $s_1 \stackrel{\tau}{\to} s_2/s_3$ then the defender makes a sequence of τ-moves where a 'check' fails and goes (via the control-state *fail*) to a configuration with control-state p_2/p_3. This is weakly bisimilar to s_2/s_3 by definition.
2. If α_2 is a configuration with control-state p_2 then this is bisimilar to s_2 by definition.
   - If the attacker makes a move $\alpha_2 \stackrel{\tau}{\to} \alpha_2'$ with $\alpha_2, \alpha_2' \in C_2$ then the defender responds by doing nothing. If the attacker makes a move $\alpha_2 \stackrel{a}{\to} \alpha_2'$ (this is only possible if the symbol A is at the top of the stack) then the control-state of α_2' is q_{verify} and $\alpha_2' \in C_1$. Thus the defender can respond by $s_2 \stackrel{a}{\to} s_1$.
   - If the attacker makes a move $s_2 \stackrel{a}{\to} s_1$ then the defender responds as follows: First he makes a sequence of τ-moves $\alpha_2 \stackrel{\tau^*}{\to} \alpha_2'$ that pops symbols

from the stack without doing any 'check' until the special symbol A is at the top. Then he makes a move $\alpha_2' \stackrel{a}{\rightarrow} \alpha_2''$. By definition the control-state of α_2'' is q_{verify} and $\alpha_2'' \in C_1$.

3. A configuration $\alpha_3 \in C_3$ can never reach a configuration where it can do action 'a'. The only possible action is τ. Thus $\alpha_3 \approx s_3$.

Since the initial configuration of P is in C_1 and the initial state of F is s_1, we get $P \approx F$. □

Theorem 4. *Weak bisimilarity of pushdown automata and finite automata is PSPACE-hard, even for the fixed finite automaton F of Figure 2.*

Proof. By reduction of the acceptance problem for single tape nondeterministic linear space-bounded Turing machines. Let M, w, P and F be defined as above. If M accepts w then by Lemma 2 P is not weakly bisimilar to any finite automaton and thus $P \not\approx F$. If M doesn't accept w then by Lemma 3 $P \approx F$. □

Theorem 5. *Weak finiteness of pushdown automata is PSPACE-hard.*

Proof. By reduction of the acceptance problem for single tape nondeterministic linear space-bounded Turing machines. Let M, w, P and F be defined as above. If M accepts w then by Lemma 2 P is not weakly bisimilar to any finite automaton and thus not weakly finite. If M doesn't accept w then by Lemma 3 $P \approx F$ and thus P is weakly finite. □

4 Hardness of Strong Bisimulation Problems

STRONG BISIMILARITY OF PUSHDOWN AUTOMATA AND FINITE AUTOMATA

Instance: A pushdown automaton P and a finite automaton F.
Question: $P \sim F$?

We show that this problem is *PSPACE*-hard in general, but polynomial in the size of P for every fixed finite automaton F. The *PSPACE* lower bound is shown by a reduction of the *PSPACE*-complete problem of quantified boolean formulae (QBF). Let $n \in \mathbb{N}$ and let $x_1, \ldots, x_n$ be boolean variables. W.r. we assume that n is even. A literal is either a variable or the negation of a variable. A clause is a disjunction of literals. The quantified boolean formula Q is given by

$$Q := \forall x_1 \exists x_2 \ldots \forall x_{n-1} \exists x_n (Q_1 \wedge \ldots \wedge Q_k)$$

where the Q_i are clauses. The problem is if Q is valid. We reduce this problem to the bisimulation problem by constructing a pushdown automaton P and a finite automaton F s.t. Q is valid iff $P \sim F$.

F is defined as follows: The initial state is s_0.

$$
\begin{array}{lll}
s_{2i} \xrightarrow{x_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} s_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
s_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{x_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
t_{2i} \xrightarrow{\bar{x}_{2i+1}} t_{2(i+1)} & \text{for } 1 \leq i \leq n/2-1 \\
s_n \xrightarrow{a} u & \\
u \xrightarrow{c} u & \\
t_n \xrightarrow{a} u & \\
t_n \xrightarrow{a} w_n & \\
w_i \xrightarrow{c} w_{i-1} & \text{for } 1 \leq i \leq n
\end{array}
$$

Note that, unlike in the previous section, the size of F is not fixed, but linear in n. Figure 3 illustrates the construction.

Now we define the pushdown automaton P. Initially the stack is empty and the initial control-state is p_0. For $1 \leq j \leq k$ and $1 \leq l \leq n$ we define $Q_j(X_l)$ iff X_l makes the clause Q_j true and $Q_j(\bar{X}_l)$ iff $\bar{X}_l$ makes Q_j true. The transitions of P are as follows:

$$
\begin{array}{lll}
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)}X_{2i+2}X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} p_{2(i+1)}\bar{X}_{2i+2}X_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)}X_{2i+2}\bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} p_{2(i+1)}\bar{X}_{2i+2}\bar{X}_{2i+1} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{x_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_{2i} \xrightarrow{\bar{x}_{2i+1}} r_{2(i+1)} & \text{for } 0 \leq i \leq n/2-1 \\
p_n \xrightarrow{a} q_j & \text{for } 0 \leq j \leq k \\
q_0 \xrightarrow{c} q_0 & \\
q_jX_l \xrightarrow{c} q_jX_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(X_l). \\
q_jX_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(X_l). \\
q_j\bar{X}_l \xrightarrow{c} q_j\bar{X}_l & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } Q_j(\bar{X}_l). \\
q_j\bar{X}_l \xrightarrow{c} q_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq n \text{ if } \neg Q_j(\bar{X}_l).
\end{array}
$$

Additionally we define for $1 \leq i \leq n/2-1$ that in the control-state r_{2i} the stack is ignored and the systems behaves just like t_{2i} in the system F of Figure 3.

Lemma 6. *If Q is not valid then $P \not\sim F$.*

Proof. If Q is not valid then $\exists x_1 \forall x_2 \ldots \exists x_{n-1} \forall x_n (\neg Q_1 \vee \ldots \vee \neg Q_k)$ and the attacker has the following winning strategy: The attacker chooses the values for the variables with the odd indices by doing actions x_i or $\bar{x}_i$ in the finite automaton F and goes from s_0 to s_n. The defender can respond in two different ways: (1) If the defender goes into a control-state r_{2i} for some i then the attacker can

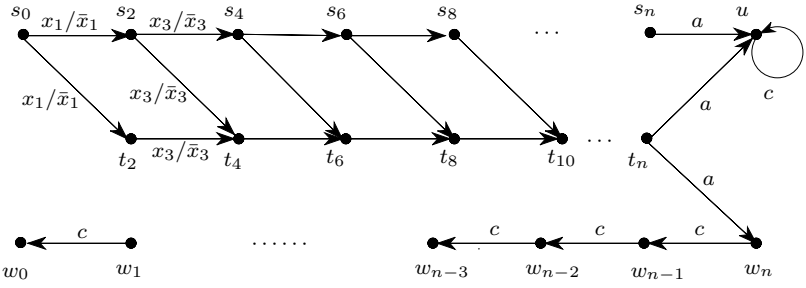


Fig. 3. Reducing QBF to strong bisimulation.

easily win, since r_{2i} behaves like t_{2i} and $s_{2i} \not\sim t_{2i}$ for every i. (2) If the defender stays in the 'p-domain' of control-states, he is forced to store the attacker's choices for the variables with odd indices on the stack. However, he can make his own choices for the variables with even indices and also stores them on the stack. Finally, the defender reaches the control-state p_n and the stack contains an assignment of values to all n variables. Since Q is not valid, there exists at least one Q_j with $1 \leq j \leq k$ that is not satisfied by this assignment. Now the attacker changes sides and makes the move $p_n \stackrel{a}{\to} q_j$ in the pushdown automaton P. The defender can only respond by making the move $s_n \stackrel{a}{\to} u$ in the system F. Now the pushdown automaton P can do the action 'c' only n times, while system F in state u can do it infinitely often. Thus the attacker can win. It follows that $P \not\sim F$. □

Lemma 7. *If Q is valid then $P \sim F$.*

Proof. Let C be a content of the stack and thus a (possibly incomplete) assignment of values to variables. Let $Q_i(C)$ be true iff C makes clause Q_i true. Let $Q(C) := \bigwedge_{1 \leq i \leq k} Q_i(C)$. Let $QX(C)$ be true iff C can be completed to a C' s.t. $Q(C')$. If Q is valid then the following relation is a strong bisimulation.

$$\{(p_{2i}C, s_{2i}) \mid 0 \leq i \leq n/2 \wedge QX(C)\} \cup \{(p_{2i}C, t_{2i}) \mid 1 \leq i \leq n/2 \wedge \neg QX(C)\} \cup \{(r_{2i}C, t_{2i}) \mid 1 \leq i \leq n/2\} \cup \{(q_jC, u) \mid 1 \leq j \leq k \wedge Q_j(C)\} \cup \{(q_0C, u)\} \cup \{(q_jC, w_i) \mid 1 \leq j \leq k \wedge 0 \leq i \leq n \wedge \neg Q_j(C) \wedge \mathit{length}(C) = i\}$$

Since $(p_0\epsilon, s_0)$ is in this relation, we get $P \sim F$. □

Theorem 8. *Strong bisimilarity of pushdown automata and finite automata is PSPACE-hard.*

Proof. Directly from Lemma 6 and Lemma 7. □

Corollary 9. *Strong bisimilarity of pushdown automata is PSPACE-hard.*

Note that Theorem 4 is not a corollary of Theorem 8. For weak bisimilarity the hardness result holds even for the small fixed finite automaton of Figure 2. However, strong bisimilarity of a pushdown automaton P and a finite automaton F is polynomial in the size of P for *every* fixed F.

Theorem 10. *Let F be a fixed finite automaton. For every pushdown automaton P the problem if $P \sim F$ requires only polynomial time in the size of P.*

Proof. Using the construction from [14] one can reduce the problem $P \sim F$ to a model checking problem in the temporal logic EF (a fragment of CTL). One can effectively construct Hennessy-Milner Logic formulae Φ and Ψ that depend only on F s.t.

$$P \sim F \iff (P \models \Phi) \wedge (P \models \neg EF\,\Psi)$$

where the modal operator EF denotes reachability. Let n be the size of (the description of) P and m the maximum of the nesting-depth of Φ and Ψ. (The total size of Φ and Ψ can be $\mathcal{O}(2^m)$.) Let P' be a state that is reachable from P. It depends only on the control state of P and P' and on the first m stack symbols of P and P' if they satisfy Φ and Ψ, respectively. There are only n different possibilities for the control state and n^m different possibilities for the first m stack symbols. For each of these n^{m+1} configurations we check if it satisfies Φ or Ψ. Each of those checks can be done in $\mathcal{O}(n^m)$ time. Also for each α of these n^{m+1} configurations we check if P can reach a configuration $\alpha\beta$ for some β. (β represents the stack contents below the first m stack symbols. It does not matter for Φ and Ψ.) Each of those (generalized) reachability-checks can be done in $\mathcal{O}(n^3m^2)$ time [3]. Therefore the whole property above can be checked in $\mathcal{O}(n^{2m+1}m^2)$ time. Thus the problem is polynomial in n, the size of P, but exponential in m. (To be precise, m depends only on F and can be made linear in the number of states in F [14].) □

Now we consider the strong finiteness problem.

Strong Finiteness of Pushdown Automata

Instance: A pushdown automaton P.
Question: Does there exist a finite automaton F s.t. $P \sim F$?

We show that this problem is *PSPACE*-hard by a reduction of QBF. Let Q, P and F be defined just as before in the hardness proof of strong bisimilarity. As shown before, Q is valid iff $P \sim F$. We now construct a pushdown automaton P' s.t. P' is finite w.r.t. strong bisimilarity iff $P \sim F$. The initial configuration of P' is $p'Z$. The transition rules are

$$\begin{aligned}
p' &\xrightarrow{a'} p'C \\
p' &\xrightarrow{a'} q' \\
q'C &\xrightarrow{b'} q' \\
q'C &\xrightarrow{c'} p_0 \\
q'Z &\xrightarrow{b'} q'Z \\
q'Z &\xrightarrow{c'} s_0
\end{aligned}$$

Note that if P' is in control-state p_0 or s_0 then it behaves like P and F, respectively.

Lemma 11. *If $P \not\sim F$ then P' is infinite w.r.t. strong bisimilarity.*

Proof. There are infinitely many non-bisimilar reachable states $q'C^iZ$ for all $i \in \mathbb{N}$. It suffices to show that $q'C^iZ \not\sim q'C^jZ$ for $i > j$. The attacker has the following winning strategy: He does action b' exactly j times (the defender can respond in only one way) and the new state in the bisimulation game is $(q'C^{i-j}Z, q'Z)$. Then the attacker does action c' and after the defender's response the new state is $(p_0C^{i-j-1}Z, s_0)$. Since $P \not\sim F$, the attacker can win. □

Lemma 12. *If $P \sim F$ then P' is finite w.r.t. strong bisimilarity.*

Proof. Let the finite automaton F' with initial state s' be defined by

$$\begin{aligned} s' &\xrightarrow{a'} s' \\ s' &\xrightarrow{a'} t' \\ t' &\xrightarrow{b'} t' \\ t' &\xrightarrow{c'} s_0 \end{aligned}$$

where s_0 is the initial state of F. If $P \sim F$ then $p'C^iZ \sim s'$, $q'C^jZ \sim t'$, $p_0C^kZ \sim s_0$ and $s_0 \sim s_0$ and thus $P' \sim F'$. □

Theorem 13. *Strong finiteness of pushdown automata is PSPACE-hard.*

Proof. It follows from Lemmas 6,7, 11 and 12 that Q is satisfiable iff $P \sim F$ iff P' is finite w.r.t. strong bisimilarity. □

It might seem that Theorem 5 is a corollary of Theorem 13. However, a careful inspection reveals a slight difference. The proof of Theorem 5 shows that the question if, given a pushdown automaton P, "Is P weakly bisimilar to any finite automaton with at most 3 states ?" is *PSPACE*-hard. The same question for strong bisimilarity is polynomial, because of Theorem 10. (These results still hold if the number 3 in the question above is replaced by any other integer $k \geq 3$. For weak bisimilarity the question is *PSPACE*-hard in the size of P. For strong bisimilarity it is polynomial in the size of P and exponential in k.) So, while in general the finiteness problem for a pushdown automaton P is *PSPACE*-hard for both weak and strong bisimilarity, the modified question "Is P finite *and small* ?" is *PSPACE*-hard for weak bisimilarity, but polynomial for strong bisimilarity. To conclude, finiteness w.r.t. weak bisimilarity is hard in a slightly stronger sense.

5 Conclusion

We have shown that all bisimulation problems for pushdown automata are at least *PSPACE*-hard. Thus no bisimulation problem for pushdown automata is polynomial (unless $PSPACE = \mathcal{P}$). It is interesting to compare these results with the results for context-free processes (BPA), which describe exactly the same class of languages (Chomsky-2). Strong and weak bisimilarity of BPA and finite automata can be decided in polynomial time [17]. This shows that there is a significant difference between pushdown automata and context-free processes (BPA) as far as 'branching-time equivalences' like strong and weak bisimulation are concerned. Intuitively, the reason for this is that, due to their finite control, pushdown automata have a limited power of self-test that context-free processes lack.

The problem of bisimulation equivalence is related to the problem of language equivalence for deterministic systems, e.g., the problem of language equivalence for deterministic pushdown automata (dPDA), which has been shown to be decidable in [26]. However, the relationship is more complex than it seems, because of the presence of ϵ-transitions in PDAs. 'Real-time' PDAs are PDAs without ϵ-transitions. We denote them by rPDA. We denote real-time deterministic PDAs as rdPDA. We can distinguish five problems.

1. For rdPDA, strong bisimilarity and trace-language equivalence coincide. (The problem of trace-language equivalence can easily be reduced to terminal-language equivalence on rdPDA.) This problem is also equivalent to strong bisimilarity of dPDA, because the ϵ-transitions don't matter for strong bisimilarity. Language equivalence on rdPDA has been shown to be decidable in [23]. Neither an upper complexity bound nor a lower complexity bound is known.
2. Strong bisimilarity for PDA and rPDA. These problems are equivalent, because the ϵ-transitions don't matter for strong bisimilarity. Decidability of strong bisimilarity for PDA has been shown in [27]. No upper complexity bound is known. Theorem 8 gives a *PSPACE* lower bound.
3. Language equivalence of dPDA. This is equivalent to weak bisimilarity of dPDA, if one renames the ϵ-transitions to τ-transitions. The problem is decidable by [26]. Neither an upper complexity bound nor a lower complexity bound is known.
4. Weak bisimilarity for PDA. It is an open question if this problem is decidable. A *PSPACE* lower bound has been shown in [28] (even for BPA). Theorem 4 shows that even the asymmetric problem of weak bisimilarity of a PDA and a (small fixed) finite automaton is *PSPACE*-hard.
5. Language equivalence for PDA and rPDA. These problems are inter-reducible and undecidable by [11].

Figure 4 shows the relationships between these five problems. The hardness results of this paper hold only for bisimilarity of nondeterministic PDA (i.e., problems number 2 and 4) and thus they don't yield a lower bound for the

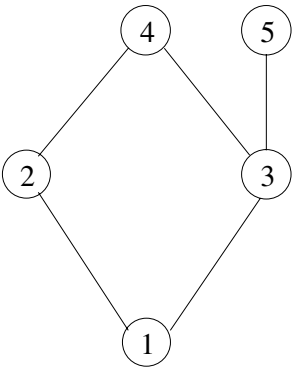


Fig. 4: Bisimulation vs. languages

problem of language equivalence of dPDA (problem number 3). In particular, it is easy to see that language equivalence of a dPDA and a deterministic finite automaton is polynomial (unlike bisimilarity for nondeterministic systems; see Theorem 8). It still cannot be ruled out that a polynomial algorithm for language equivalence of dPDA might exist.

Two lower bounds for bisimulation problems about Petri nets have not been mentioned explicitly in the literature so far. They concern the problems of strong bisimilarity of a Petri net and a finite automaton and finiteness of a Petri net w.r.t. strong bisimulation. It can easily be shown that these problems are *EXPSPACE*-hard by a reduction of the problem if a given place in a Petri net can ever become marked. (This problem is polynomially equivalent to the reachability problem for Petri nets [25] and thus *EXPSPACE*-hard [18].)

Table 1 summarizes known results about the complexity of bisimulation problems for several classes of infinite-state systems. The different columns show the results about the following problems: strong bisimilarity with finite automata, strong bisimilarity of two infinite-state systems, weak bisimilarity with finite automata and weak bisimilarity of two infinite-state systems. New results are in boldface.

Table 1.

	$\sim F$	$\sim$	$\approx F$	$\approx$
FS	$\mathcal{P}$ [2,24]	$\mathcal{P}$ [2,24]	$\mathcal{P}$ [2,24]	$\mathcal{P}$ [2,24]
BPA	$\mathcal{P}$ [17]	$\in 2\text{-}EXPTIME$ [4]	$\mathcal{P}$ [17]	*PSPACE*-hard [28]
PDA	$\in$ *EXPTIME* [14] **PSPACE-hard**	decidable [27] **PSPACE-hard**	$\in$ *EXPTIME* [14] **PSPACE-hard**	*PSPACE*-hard [28]
BPP	$\in$ *PSPACE* [14]	decidable [7] co-$\mathcal{NP}$-hard [19]	$\in$ *PSPACE* [14]	$\mathcal{NP}$-hard [28] Π_2^p-hard [19]
PA	decidable [14]	co-$\mathcal{NP}$-hard [19]	decidable [14]	*PSPACE*-hard [28]
PAD	decidable [14] **PSPACE-hard**	**PSPACE-hard**	decidable [14] **PSPACE-hard**	*PSPACE*-hard [28]
PN	decidable [15,14] *EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PAN	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]
PRS	*EXPSPACE*-hard	undecidable [12]	undecidable [12]	undecidable [12]

Table 2 summarizes results about the problems of strong and weak finiteness. New results are in boldface.

Table 2.

	strong finiteness	weak finiteness
BPA	$\in 2\text{-}EXPTIME$ [5,4]	?
PDA	**PSPACE-hard**	**PSPACE-hard**
BPP	decidable [13] co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PA	co-$\mathcal{NP}$-hard [19]	Π_2^p-hard [19]
PAD	**PSPACE-hard**	**PSPACE-hard**
PN	decidable [13] *EXPSPACE*-hard	undecidable [13]
PAN/PRS	*EXPSPACE*-hard	undecidable [13]

Some more results are known about the restricted subclasses of these systems that satisfy the 'normedness condition' (e.g. [10,9,8,16]). Normedness means that from every reachable state there is a terminating computation. This condition makes many bisimulation problems much easier, e.g., strong bisimilarity of normed BPP is decidable in polynomial time [10], while it is at least co-$\mathcal{NP}$-hard in the general case [19]. Also for normed systems finiteness w.r.t. strong bisimilarity coincides with boundedness [16], while this doesn't hold in the general case.

Acknowledgment: Thanks to Colin Stirling for helpful discussions.

References

[1] J.C.M. Baeten and W.P. Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[2] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[3] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[4] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[5] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[6] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.

[7] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.

[8] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.
[9] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.
[10] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.
[11] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.
[12] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.
[13] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.
[14] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.
[15] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.
[16] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.
[17] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.
[18] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
[19] R. Mayr. On the complexity of bisimulation problems for Basic Parallel Processes. In *Proc. of ICALP'2000*, volume ? of *LNCS*. Springer Verlag, 2000.
[20] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
[21] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[22] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.
[23] M. Oyamaguchi, N. Honda, and Y. Inagaki. The equivalence problem for real-time strict deterministic languages. *Information and Control*, 45:90–115, 1980.
[24] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
[25] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.
[26] G. Sénizergues. The Equivalence Problem for Deterministic Pushdown Automata is Decidable. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 671–681. Springer Verlag, 1997.
[27] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS'98*. IEEE, 1998.
[28] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.

A Type-Theoretic Study on Partial Continuations

Yukiyoshi Kameyama

Graduate School of Informatics, Kyoto University 606-8501, JAPAN,
kameyama@kuis.kyoto-u.ac.jp

Abstract. *Partial continuations* are control operators in functional programming such that a function-like object is abstracted from a part of the rest of computation, rather than the whole rest of computation. Several different formulations of partial continuations have been proposed by Felleisen, Danvy&Filinski, Hieb et al, and others, but as far as we know, no one ever studied logic for partial continuations, nor proposed a typed calculus of partial continuations which corresponds to a logical system through the Curry-Howard isomorphism. This paper gives a simple type-theoretic formulation of a form of *partial continuations* (which we call delimited continuations), and study its properties. Our calculus does reflect the intended operational semantics, and enjoys nice properties such as subject reduction and confluence. By restricting the type of delimiters to be atomic, we obtain the normal form property. We also show a few examples.

1 Introduction

The mechanism of first-class continuations (the `call/cc`-mechanism in Scheme [1]) is a quite powerful control facility, and is equipped with many modern programming languages such as Standard ML. Felleisen et al [5] established a theory for first-class continuations by which we can reason about properties of programs with first-class continuations.

Partial continuation is a refinement of first-class continuation in that a continuation object is abstracted from a part of the rest of computation, rather than the whole rest of computation. Felleisen [6] introduced a pair of operators # and $\mathcal{F}$ to represent partial continuations. The former delimits the range of continuations which will be later invoked by the latter operator. The other distinguished feature of partial continuations is that, its invocation does not abort the current continuation, contrary to the first-class continuations. Hence the abstracted objects are normal functions which can be composed with other functions. As Felleisen showed, the concept of partial continuation is useful in practice; interesting examples can be implemented more concisely and efficiently using partial continuations. After then, several different operators for partial continuations have been proposed by Queinnec and Serpette [20], Gunter [11], Danvy and Filinski [2] and others.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 489–504, 2000.
© Springer-Verlag Berlin Heidelberg 2000

If we want to give a logical view to partial continuations through the Curry-Howard isomorphism, a fundamental problem arises in these approaches. Namely, the scope of Felleisen's # operator (and its counterpart in other researcher's calculi) is dynamic; for each $\mathcal{F}$ operator, the matching # is determined at the time of evaluation. Consequently, we cannot represent the scope of # by a simple variable-binding mechanism, thus cannot construct a simple logic corresponding to their operators. One exception is the subcontinuation by Hieb et al [13] whose operator has static scope. However, their operator may generate run-time errors and we cannot develop a type safe calculus for subcontinuations.

In this paper, we give a simple typed calculus for partial continuations which have static scope. Since the scope of a partial continuation is lexically determined by the corresponding delimiter, our variant is called a delimited continuation.[1] Our calculus is designed to satisfy the following conditions: (1) it is a statically typed system which corresponds to, through the Curry-Howard isomorphism, a consistent logical system, (2) it is type safe in the sense that it enjoys the subject reduction property and reductions never get stuck, and (3) its reduction rules are confluent, and compatible with any contexts. By (1), our calculus can be viewed as a logical system. Indeed, it corresponds to classical logic. By (2) and (3), we have an equational theory for programs with partial continuations.

In order to make our type-theoretic analysis easier, we represent (the counterpart of) $\mathcal{F}$ by two operators, an invoker of partial continuations and a throw operation, and give reduction rules. Our reduction rules reflect the intended operational semantics, and enjoy the above properties (1)-(3). We also show that the subcalculus with the delimiter and the throw operation (without the invoker) corresponds to the classical catch/throw calculus in [18,14], while the subcalculus with the delimiter and the invoker (without the throw operation) corresponds to intuitionistic calculus.

The rest of this paper is organized as follows. Section 2 gives the background and motivation of our formulation. Sections 3, 4 and 5 give the type system, the operational semantics, and the reduction rules of our calculus, respectively. Section 6 proves theoretical properties such as subject reduction and confluence. Section 7 shows that our calculus corresponds to classical logic through the Curry-Howard isomorphism. Section 8 gives the conclusion.

2 Formulating Partial Continuations in Type Theory

We start our analysis with Felleisen's formulation. Felleisen [6] proposed a form of partial partial continuations, which has the following reduction rule:

$$\#E[\mathcal{F}V] \to V(\lambda x.E[x])$$

where E is an evaluation context, V is a value, # is a delimiter which restricts the range of partial continuations, and $\mathcal{F}$ is an operator which creates a partial-continuation object up to the delimiter. In the above term, the created partial

[1] Olivier Danvy coined this term.

continuation object $\lambda x.E[x]$ is applied to the term V. Note that, this object is a simple function, and not abortive in the sense that, when some value is applied, it does not throw away the current continuation. These two features are characteristic points for partial continuations compared with first-class (full) continuations.

Felleisen's partial continuations are refinement of full continuations; theoretically partial continuations behaves well, and are useful in practice. However, if we want to construct a typed calculus for partial continuations which corresponds to a sound logical system, it is problematic.

The problem is that, the scope of the #-operator is dynamic, for instance, $(\lambda x.\#(xN))(\lambda y.\mathcal{F}M)$ reduces to $\#((\lambda y.\mathcal{F}M)N)$, thus $\mathcal{F}$ gets captured by # through this reduction. In this system, we cannot intuitively understand the meaning of programs. Consider the term $\lambda x.\#(x(\mathcal{F}M))$. We are tempted to consider this # and $\mathcal{F}$ are connected. But if we apply $\lambda y.C[\#y]$ to it, the above $\mathcal{F}$ is delimited by the latter #. It seems impossible to have a Curry-Howard isomorphism of this kind of calculi and ordinary logical systems, since the correspondence between the #-operator and the $\mathcal{F}$-operator cannot be represented by the variable-binding mechanism (which is lexical).

Danvy and Filinski proposed another formulation of partial continuations [2] [3]. Their operators `reset` and `shift` differ from Felleisen's ones in that the created partial continuation object is again delimited. This change has a better effect for formalizing partial continuations, and in fact, they successfully gave a CPS-translation in an ordinary, functional style. However, still the scope of their `reset` operator (denoted by # in the above reduction) is also dynamic, hence the same problem applies if one want to formalize their operator in a type theory which admits the Curry-Howard isomorphism.

Hieb, Dybvig and Anderson [13] proposed subcontinuations which essentially have the following reduction rule:

$$\#_l(E[\mathcal{F}_l V]) \to V(\lambda x.\#_l(E[x]))$$

where l is a label attached to the operators. The notable point in their approach is that (i) they can treat multiple labels so that the operator $\mathcal{F}$ can specify the matching delimiter, and (ii) the binding mechanism of the label l is the ordinary variable binding so that it is static. These two points are big benefits for logical viewpoint. However, the $\mathcal{F}$-operator may become unbound through the reduction, causing a run-time type-error (when V contains an occurrence of the label l, then the reduced term may contain l free). Also they did not study a typed calculus for their operator.

Our aim is to develop a theory for partial continuations which is logically well-founded. Since existing partial continuation operators are not satisfactory in the sense of logic, we shall change the operational behavior of existing partial continuations to obtain a logically well-behaved system. We should be careful for this change of semantics so that interesting programming examples with existing partial continuations can be expressible in our calculus. In particular, we should try to make this change as little as possible.

As a conclusion we decided to formalize an improved version of Hieb et al's operator so that no run-time type error may occur. In order to make our analysis easier, we shall not directly formalize the $\mathcal{F}$-operator, but instead we treat two operators `calldc` (stands for "call with partial continuation") and `throw`, which essentially have the following reductions:

$$\#_\alpha E[\texttt{calldc}_\alpha V] \to \#_\alpha E[V(\lambda x.\#_\alpha E[x])]$$

$$\#_\alpha E[\texttt{throw}_\alpha V] \to \#_\alpha V$$

where V is a value and E is an evaluation context defined later. The point is that, in the first rule we attach one more delimiter to enclose the resulting term. As is shown later, our (counterpart of the) $\mathcal{F}$-operator can be represented by `calldc` and `throw`.

3 The Type System

We now define the type system of our calculus. Actually we are defining two calculi $\lambda_{\texttt{DC}}$ and $\lambda_{\texttt{DC}}^{\text{atomic}}$. $\lambda_{\texttt{DC}}$ is the full calculus, and by putting restriction on terms we obtain $\lambda_{\texttt{DC}}^{\text{atomic}}$.

Types and terms are defined by the following grammar where K is an atomic type, c^K is a constant of atomic type K, and x and α are variables[2]. We assume that `Unit` is an atomic type, and $\bullet$ is a constant (its single element) of type `Unit`.

$$\begin{aligned} A, B &::= K \mid A \to B \mid \neg A \\ M, N &::= x \mid c \mid \lambda x.M \mid MN \\ &\quad \mid \#_\alpha M \mid \texttt{calldc}_\alpha M \mid \texttt{throw}_\alpha M \end{aligned}$$

The type $\neg A$ is the type of tags of control operators for type A. Note that $\neg$ is a primitive type constructor, and not a defined symbol. In $\lambda_{\texttt{DC}}^{\text{atomic}}$, we restrict that the types of tags be atomic, namely in formulating $\neg A$, the type A must be atomic. $\lambda_{\texttt{DC}}$ has no restriction.

The first line of terms are usual λ-terms. The second line consists of control operators. The term $\#_\alpha M$ delimits the scope of partial continuation which may be created inside M (with the tag α). The term $\texttt{calldc}_\alpha M$ creates a partial-continuation (delimited continuation) object like $\mathcal{F}$-operator. Our calculi also have an abortive operator $\texttt{throw}_\alpha M$ which finishes the current continuation up to the corresponding delimiter. As a notational convention, $\#_\alpha M_1 \ldots M_n$ should be parsed as $\#_\alpha(M_1 \ldots M_n)$.

A type environment is a finite set of the form $x : A$ where no variable appears more than once. For instance $\{x : A,\ \alpha : \neg(B \to C)\}$ is a type environment.

[2] There is no syntactic distinction of x and α, but we use x for ordinary variable and α for tags.

We use Γ for a type environment. A judgement is in the form $\Gamma \vdash M : A$. The typing rules to derive a judgement are displayed in Table 1. As usual, if two type environments Γ_1 and Γ_2 are not compatible, then $\Gamma_1 \cup \Gamma_2$ is not defined. If $\Gamma \vdash M : A$ is derived, we say M is a (typable) term of type A under the type environment Γ. We sometimes omit type environments if they are apparent.

Table 1. Typing Rules

$$\frac{}{\Gamma \cup \{x : A\} \vdash x : A} \qquad \frac{}{\Gamma \vdash c^K : K}$$

$$\frac{\Gamma \cup \{x : A\} \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma_1 \vdash M : A \to B \quad \Gamma_2 \vdash N : A}{\Gamma_1 \cup \Gamma_2 \vdash MN : B}$$

$$\frac{\Gamma \cup \{\alpha : \neg B\} \vdash M : B}{\Gamma \vdash \#_\alpha M : B}$$

$$\frac{\Gamma \vdash M : ((\mathtt{Unit} \to A) \to B) \to A}{\Gamma \cup \{\alpha : \neg B\} \vdash \mathtt{calldc}_\alpha M : A} \qquad \frac{\Gamma \vdash M : B}{\Gamma \cup \{\alpha : \neg B\} \vdash \mathtt{throw}_\alpha M : C}$$

In Table 1, The first three rules are as usual. The fourth rule is for the control delimiter, and as we explained, it discharges the assumption $\alpha : \neg B$, namely, it binds the variable α.

The fifth and sixth rules are for `calldc` and `throw`. The role of `calldc` is almost the same as $\mathcal{F}$, but its type is slightly different, since (i) we split the role of $\mathcal{F}$ into two operators `calldc` and `throw`, and (ii) by controlling the evaluation order suitably, we used a *thunk* of type $\mathtt{Unit} \to A$ in the rule for `calldc`. We can delay the evaluation of a term M of type A by encapsulating it as $\lambda x^{\mathtt{Unit}}.M$. This technique is useful when we argue the correspondence between the ML-like operational semantics and this formulation in Section 6. Since $\mathtt{throw}_\alpha M$ aborts the current continuation and jumps to the corresponding control-delimiter, its type can be any type. The variable α in these two operators is a free occurrence.

The variables x and α are bound by $\lambda x.M$ and $\#_\alpha M$, respectively, and $FV(M)$ denotes the set of free variables in M. As usual, we identify two terms which differ in bound variables only. The term $M[N/x]$ denotes the term M substituted N for x. We also say that a term M is closed if $FV(M) = \{\}$.

Note that α is an ordinary variable, so we may abstract α like $\lambda\alpha.M$. This has practical benefit, since we often want to define functions with free tags, and bind them in another function. Let us show an example in Scheme-like language. We often want to write the following program:

```
(define (foo x)
   ...   (callpc alpha ...) ...)

(define (goo y)
   (catch alpha (foo y)))
```

In our calculus, such a program is not allowed (since our control operators have static scope). However, we can represent the above program by abstracting the variable `alpha` in `foo`. The resulting program is:

```
(define (foo x beta)
   ...   (callpc beta ...) ...)

(define (goo y)
   (catch alpha (foo y alpha)))
```

By this technique, we can (partly) recover the expressiveness, which was lost by changing the scope of delimiters from dynamic one to static one. However, not all expressible programs with dynamic scope operators can be expressed in our calculus. We think that it is a trade-off of theory and practice.

4 ML-Like Operational Semantics

In this section we give an operational semantics of our calculi in the style of Felleisen et al [8]. Note that this operational semantics may cause run-time type errors, therefore its direct formalization (in a type safe way) is not possible. We nevertheless state the operational semantics here to clarify our intented semantics. Later we show that the equality in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ corresponds to this operational semantics.

We first define values (denoted by V), redexes (denoted by R), and evaluation contexts (denoted by E) as follows:

$$
\begin{aligned}
V &::= x \mid c \mid \lambda x.M \\
R &::= (\lambda x.M)V \mid \mathtt{\#}_\alpha V \mid \mathtt{calldc}_\alpha V \mid \mathtt{throw}_\alpha V \\
E &::= [\,] \mid EM \mid VE \mid \mathtt{calldc}_\alpha E \mid \mathtt{throw}_\alpha E \mid \mathtt{\#}_\alpha E
\end{aligned}
$$

Then we have that, any closed term M is either a value, or there uniquely exists a pair of a redex R and an evaluation context E such that $M \equiv E[R]$.

We have the following 1-step reduction rules where E_0 is an evaluation context which does not bind α.

$$
\begin{aligned}
E[(\lambda x.M)V] &\rightsquigarrow_1 E[M[V/x]] \\
E[\mathtt{\#}_\alpha V] &\rightsquigarrow_1 E[V] \\
E[\mathtt{\#}_\alpha E_0[\mathtt{calldc}_\alpha V]] &\rightsquigarrow_1 E[\mathtt{\#}_\alpha E_0[V(\lambda u.\mathtt{\#}_\alpha E_0[u\bullet])]]
\end{aligned}
$$

$$E[\#_\alpha E_0[\mathtt{throw}_\alpha V]] \rightsquigarrow_1 E[V]$$
$$E_0[\mathtt{calldc}_\alpha V] \rightsquigarrow_1 \text{error}$$
$$E_0[\mathtt{throw}_\alpha V] \rightsquigarrow_1 \text{error}$$

Since the decomposition of a term is unique, the above set of reduction rules induces a deterministic evaluation strategy.

Run-time errors may happen for $\lambda_{\mathtt{DC}}$ even if a reduction begins with a closed term; in the second and the fourth rules, the value V may be $\lambda x.M$ and M may contain free occurrences of α. For instance:

$$(\#_\alpha \lambda x.\mathtt{calldc}_\alpha x)V \rightsquigarrow_1 (\lambda x.\mathtt{calldc}_\alpha x)V \rightsquigarrow_1 \mathtt{calldc}_\alpha V \rightsquigarrow_1 \text{error}$$

On the contrary, no run-time errors occur in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$, since the value V in $\#_\alpha V$ and $\mathtt{throw}_\alpha V$ must be either a variable or a constant.

5 Small-Step Reductions

We want to set up an equational theory to reason about the programs in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$. In this respect the reduction step given in the previous section is too big. This section gives finer reduction rules which are easier to study.

First we define a singular context E_s as follows:

$$E_s ::= [\,]M \mid V[\,] \mid \mathtt{calldc}_\alpha[\,] \mid \mathtt{throw}_\alpha[\,]$$

Next we define the notion of one-step reduction denoted by $\rightarrow_1$, which is defined as the compatible closure of the following reduction rules.

We split the reduction rules into two groups. The first group of reduction rules are as follows:

$$(\lambda x.M)V \rightarrow_1 M[V/x] \tag{1}$$
$$\#_\alpha M \rightarrow_1 M \qquad (\text{if } \alpha \notin FV(M)) \tag{2}$$
$$\#_\alpha \#_\beta M \rightarrow_1 \#_\alpha M[\alpha/\beta] \tag{3}$$

The first one is the usual call-by-value β-reduction. The second one means that if there are no control operators with tag α, then the delimiter is useless. The last one means that, if two delimiters are set in the same place, they can be unified.

The second group of reduction rules are as follows:

$$\#_\alpha(\mathtt{calldc}_\alpha V) \rightarrow_1 \#_\alpha(V(\lambda u.\#_\alpha u\bullet)) \tag{4}$$
$$\#_\alpha(\mathtt{throw}_\alpha V) \rightarrow_1 \#_\alpha V \tag{5}$$
$$E_s[\mathtt{calldc}_\alpha V] \rightarrow_1 \mathtt{calldc}_\alpha(\lambda x.E_s[V(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))]) \tag{6}$$
$$\#_\beta(\mathtt{calldc}_\alpha \lambda x.M) \rightarrow_1 \mathtt{calldc}_\alpha \lambda x.\#_\beta M \qquad (if\ \alpha \not\equiv \beta) \tag{7}$$
$$E_s[\mathtt{throw}_\alpha V] \rightarrow_1 \mathtt{throw}_\alpha V \tag{8}$$

In these reductions we assume x, y, z are fresh variables.

The first two rules express reductions with an empty partial continuation. The third and fourth rules are one-step reductions for `calldc` and `throw`.

Although the righthand side of the third rule (Rule (6)) may look quite complex, it is similar to Felleisen's one-step reduction rule for first-class continuations. A crucial point of this reduction rule is that, in the righthand side, the partial continuation object is delimited by a newly introduced delimiter $\#_\alpha$, so the occurrences of $\texttt{calldc}_\alpha$ in E_s are bound by this new $\#_\alpha$. If we do not have a new delimiter in the reduct, we cannot simulate many interesting reductions which were written with partial continuations of dynamic scope.

Let $\rightarrow$ be a reflexive, transitive closure of $\rightarrow_1$, and $=$ be the least equivalence relation which contains $\rightarrow$.

Representing the $\mathcal{F}$-like operator

The `calldc` operator does not discard the current continuation. But we can define an operator for creating partial continuation objects, which discards the current continuation. Let us define `control` as follows:

$$\texttt{control}_\alpha M \stackrel{\triangle}{=} \texttt{calldc}_\alpha(\lambda x.\texttt{throw}_\alpha M x)$$

Then this operator has the following reduction in the ML-like operational semantics as desired.

$$E[\texttt{\#}_\alpha E_0[\texttt{control}_\alpha V]] \rightsquigarrow E[\texttt{\#}_\alpha V(\lambda u.\texttt{\#}_\alpha E_0[u\bullet])]$$

The `control` operator is closer to Felleisen's $\mathcal{F}$-operator and Danvy and Filinski's `shift` operator.

A Small Example

The following example was given by Danvy and Filinski [2]. In order to express this example, we assume that $+$ and its reductions were added to our calculus.

$$\begin{aligned}
& 1 + \texttt{\#}_\alpha(10 + (\lambda x.\texttt{control}_\alpha(\lambda k.k(k(x))))100) \\
\rightarrow\ & 1 + \texttt{\#}_\alpha(10 + \texttt{control}_\alpha(\lambda k.k(k(100)))) \\
\equiv\ & 1 + \texttt{\#}_\alpha(10 + \texttt{calldc}_\alpha(\lambda y.\texttt{throw}_\alpha(\lambda k.k(k(100)))y)) \\
\rightarrow\ & 1 + \texttt{\#}_\alpha(10 + (\lambda y.\texttt{throw}_\alpha(y(y(100))))(\lambda u.\texttt{\#}_\alpha 10 + u)) \\
\rightarrow\ & 1 + \texttt{\#}_\alpha\texttt{throw}_\alpha((\lambda u.10 + u)((\lambda u.10 + u)(100))) \\
\rightarrow\ & 1 + \texttt{\#}_\alpha 120 \\
\rightarrow\ & 121
\end{aligned}$$

A Note on Control Operators with Dynamic Scope

As we explained, our control operators have static scopes. If we would change them to have dynamic scopes in $\lambda_{\mathtt{DC}}$, we would have a non-terminating reduction as follows.

Let P be $\lambda x.(\#_\alpha(\lambda y.z)(xw))x$, and Q be $\lambda u.\mathtt{throw}_\alpha P$. Then, PQ has type $(C \to A) \to B$ under the type environment $\{z : (C \to A) \to B,\ w : C\}$. If the delimiter has dynamic scope, the reduction sequence from PQ does not terminate as follows:

$$\begin{aligned} PQ &\to (\#_\alpha((\lambda y.z)(Qw)))Q \\ &\to (\#_\alpha((\lambda y.z)(\mathtt{throw}_\alpha P)))Q \\ &\to (\#_\alpha \mathtt{throw}_\alpha P)Q \\ &\to (\#_\alpha P)Q \\ &\to PQ \end{aligned}$$

We do not have this reduction sequence with static scope operators, since we must rename α in P before substituting Q for x in P.

$$\begin{aligned} PQ &\to (\#_\beta((\lambda y.z)(Qw)))Q \\ &\to (\#_\beta((\lambda y.z)(\mathtt{throw}_\alpha P)))Q \\ &\to (\#_\beta \mathtt{throw}_\alpha P)Q \\ &\to (\mathtt{throw}_\alpha P)Q \\ &\to \mathtt{throw}_\alpha P \end{aligned}$$

The term $\mathtt{throw}_\alpha P$ cannot be reduced any further.

6 Properties of our Calculus

This section gives properties of $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$.

First of all, the reductions are closed under substitution.

Theorem 1. *If $M \to M'$ and $N \to N'$, then $M[N/x] \to M'[N'/x]$.*

This is obvious since our reduction rules are compatible with arbitrary contexts. Note that the calculus for the full continuations contains so called a computation rule which is not compatible with contexts. For instance $\mathcal{C}M \to M(\lambda x.\mathcal{A}x)$ is only applicable for the top-level context where $\mathcal{C}$ is Felleisen's control operator, and $\mathcal{A}$ is the abort operator.

The subject reduction property expresses the type safety.

Theorem 2 (Subject Reduction Property). *If $\Gamma \vdash M : A$ and $M \to N$, then we have that $\Gamma \vdash N : A$ and $FV(M) \subseteq FV(N)$.*

Proof. We only verify the most complex reduction rule (6). Suppose that E_s has type C, and its hole has type A. The lefthand-side term is typed as follows (omitting the type environment for readability):

$$\dfrac{\dfrac{\vdots}{M : ((\mathtt{Unit} \to A) \to B) \to A}}{\dfrac{\mathtt{calldc}_\alpha M : A}{E_s[\mathtt{calldc}_\alpha M] : C}}$$

Then, the righthand-side term is typed as follows:

$$\dfrac{\dfrac{\dfrac{\vdots}{M : ((\mathtt{Unit} \to A) \to B) \to A} \quad \dfrac{\dfrac{x : (\mathtt{Unit} \to C) \to B \quad \dfrac{\dfrac{y : \mathtt{Unit} \to A \quad \bullet : \mathtt{Unit}}{y\bullet : A}}{E_s[y\bullet] : C}}{\lambda z.E_s[y\bullet] : \mathtt{Unit} \to C}}{x(\lambda z.E_s[y\bullet]) : B}}{\#_\alpha x(\lambda z.E_s[y\bullet]) : B}}{\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]) : (\mathtt{Unit} \to A) \to B}}{M(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet])) : A}}{E_s[M(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))] : C}}{\lambda x.E_s[M(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))] : ((\mathtt{Unit} \to C) \to B) \to C}}{\mathtt{calldc}_\alpha(\lambda x.E_s[M(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))]) : C}$$

We also have that the set of free variables are the same.

Other cases are proved easily. □

We then show that $\lambda_{\mathtt{DC}}$ and $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ are confluent by using Takahashi's parallel reduction method [19] in conjunction with Hardin's interpretation method.

Theorem 3. *The calculi $\lambda_{\mathtt{DC}}$ and $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ are confluent.*

Proof. We first define a d-normal form $d(M)$ of a term M as the term M where the reduction (3) (unification of delimiters) is applied as many times as possible, namely, contracting successive application of delimiters. Apparently, for each term, its d-normal form is unique up to renaming of bound variables.

We then define the parallel reduction $\Rightarrow$ on terms as follows:

- $x \Rightarrow x$, and $c \Rightarrow c$.
- If $M_i \Rightarrow M_i'$ for $i = 1, 2$, then $\lambda x.M_1 \Rightarrow \lambda x.M_1'$, $M_1 M_2 \Rightarrow M_1' M_2'$, $\mathtt{calldc}_\alpha M_1 \Rightarrow \mathtt{calldc}_\alpha M_1'$, $\mathtt{throw}_\alpha M_1 \Rightarrow \mathtt{throw}_\alpha M_1'$, and $\#_\alpha M_1 \Rightarrow \#_\alpha M_1'$.
- If $M \Rightarrow M'$, and $V \Rightarrow V'$, then $(\lambda x.M)V \Rightarrow M'[V'/x]$.
- If $M \Rightarrow M'$ and $\alpha \notin FV(M)$, then $\#_\alpha M \Rightarrow M'$.
- If $V \Rightarrow V'$, then $\#_\alpha \mathtt{calldc}_\alpha V \Rightarrow \#_\alpha V'(\lambda u.\#_\alpha u\bullet)$.
- If $V \Rightarrow V'$, then $\#_\alpha \mathtt{throw}_\alpha V \Rightarrow \#_\alpha V'$.
- If $V \Rightarrow V'$, $E_s \Rightarrow E_s'$, then $E_s[\mathtt{calldc}_\alpha V] \Rightarrow \mathtt{calldc}_\alpha(\lambda x.E_s'[V'(\lambda y.\#_\alpha x(\lambda z.E_s'[y\bullet]))])$.

– If $V \Rightarrow V'$, then $E_s[\mathtt{throw}_\alpha V] \Rightarrow \mathtt{throw}_\alpha V'$.
– If $M \Rightarrow M'$, then $\mathtt{\#}_\beta \mathtt{calldc}_\alpha \lambda x.M \Rightarrow \mathtt{calldc}_\alpha \lambda x.\mathtt{\#}_\beta M'$.
– If $M \Rightarrow M'$, then $M \Rightarrow d(M')$.

Next, we define the term M^* for each term M as follows. In the following definition, if more than one clause match the term M, then we always take the first matching clause as the definition of M^*.

– $x^* \equiv x$, and $c^* \equiv c$.
– $(\lambda x.M)^* \equiv \lambda x.M^*$,
– $((\lambda x.M)V)^* \equiv d(M^*[V^*/x])$.
– $(E_s[\mathtt{calldc}_\alpha V])^* \equiv d(\mathtt{calldc}_\alpha \lambda x.E_s^*[V^*(\lambda y.\mathtt{\#}_\alpha x(\lambda z.E_s^*[y\bullet]))])$.
– $(E_s[\mathtt{throw}_\alpha V])^* \equiv \mathtt{throw}_\alpha V^*$.
– $(MN)^* \equiv M^* N^*$.
– $(\mathtt{calldc}_\alpha M)^* \equiv \mathtt{calldc}_\alpha M^*$, and $(\mathtt{throw}_\alpha M)^* \equiv \mathtt{throw}_\alpha M^*$.
– $(\mathtt{\#}_\alpha M)^* \equiv M^*$ if $\alpha \notin FV(M)$.
– $(\mathtt{\#}_\alpha \mathtt{\#}_\beta M)^* \equiv (\mathtt{\#}_\beta M)^*[\alpha/\beta]$.
– $(\mathtt{\#}_\alpha \mathtt{calldc}_\alpha V)^* \equiv \mathtt{\#}_\alpha(V^*(\lambda u.u\bullet))$.
– $(\mathtt{\#}_\alpha \mathtt{throw}_\alpha V)^* \equiv d(\mathtt{\#}_\alpha V^*)$.
– $(\mathtt{\#}_\beta \mathtt{calldc}_\alpha \lambda x.M)^* \equiv d(\mathtt{calldc}_\alpha \lambda x.\mathtt{\#}_\beta M^*)$.
– $(\mathtt{\#}_\alpha M)^* \equiv d(\mathtt{\#}_\alpha M^*)$.

Then by case-analysis, we have that if $M \Rightarrow N$ then $N \Rightarrow M^*$, which implies the diamond property of $\Rightarrow$. In this proof, the only problematic case is $\mathtt{\#}_\gamma \mathtt{\#}_\beta \mathtt{calldc}_\alpha \lambda x.M \Rightarrow \mathtt{\#}_\gamma \mathtt{calldc}_\alpha \lambda x.\mathtt{\#}_\beta M$, but it can be probed by some calculation.

We also have, $M \to N$ implies $M \Rightarrow N$ and then $\to$ is confluent.

Note that these arguments apply to both $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$. □

Subject reduction and confluence are most basic properties of the typed calculi, but by restricting the tag types to be atomic, we have a few more desirable properties.

Theorem 4 (Normal Form Property). *Let M be a closed normal term in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$. Then M is either a constant or in the form of $\lambda x.M'$.*

Proof. We say a term M is half-closed if $FV(M) = \{x_1, \cdots, x_n\}$ and the type of x_i is $\neg A_i$ for $1 \leq i \leq n$. We can show by induction that, a half-closed normal term M is in the following forms:

$$x_i, \quad c, \quad \lambda x.M, \quad \mathtt{calldc}_\alpha V, \quad \text{or} \quad \mathtt{throw}_\alpha V$$

The point here is that the type $\neg A$ is not defined as $A \supset \bot$, in which case $x_i c$ may be a half-closed normal term.

It follows that, if M is a closed normal term, it is a constant or a λ-abstract. □

The normal form property together with the subject reduction property ensures the type soundness for $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$.

Unfortunately, the normal form property does not hold for $\lambda_{\mathtt{DC}}$; there is a closed normal term of the form $\#_\alpha \lambda x.M$ in $\lambda_{\mathtt{DC}}$. To obtain the property in $\lambda_{\mathtt{DC}}$, we should add reduction rules such as $(\#_\alpha V_1)V_2 \to_1 \#_\alpha V_1'$ where V_1' is obtained by appropriate substitution. However this single reduction rule does not suffice, and we should add more and more. We believe that, even under the restriction of $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$, we can express many programming examples, since we usually do not want to place delimiters for function types.

We finally show that our small step reductions do characterize the operational semantics given earlier.

Theorem 5. *Let M and N be closed terms. If $M \rightsquigarrow N$, then $M = N$ in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$.*

Proof. The theorem is proved by induction on the length of evaluation. The base case is trivial. We list here only mojor cases for the induction step.

Case-1. $E[(\lambda x.M)V] \rightsquigarrow E[M[V/x]]$.

The same reduction is included in $\to_1$.

Case-2. $E[\#_\alpha V] \rightsquigarrow E[V]$ where $\alpha \notin FV(V)$.

Since we restricted the type of α be of the form $\neg K$ where K is atomic, the term V must be a constant of that type. We have $\#_\alpha c \to_1 c$ in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$.

Case-3. $E[\#_\alpha E_0[\mathtt{calldc}_\alpha V]] \rightsquigarrow E[\#_\alpha E_0[V(\lambda u.\#_\alpha E_0[u\bullet])]]$.

We use one more induction on the structure of E_0 to prove this case.

Case-3-1. If $E_0 \equiv [\ ]$ then, $E[\#_\alpha \mathtt{calldc}_\alpha V] \to_1 E[\#_\alpha V(\lambda u.\#_\alpha u\bullet)]$, which is $E[\#_\alpha E'[V(\lambda u.\#_\alpha u\bullet)]]$.

Case-3-2. If $E_0 \equiv F[E_s]$ where F is an evaluation context and E_s is a singular context, then we have

$$\begin{aligned}
\#_\alpha F[E_s[\mathtt{calldc}_\alpha V]] &\to_1 \#_\alpha F[\mathtt{calldc}_\alpha \lambda x.E_s[V(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))]] \\
&= \#_\alpha F[(\lambda x.E_s[V(\lambda y.\#_\alpha x(\lambda z.E_s[y\bullet]))])(\lambda u.\#_\alpha F[u\bullet])] \\
&\qquad \text{(by induction hypothesis)} \\
&\to \#_\alpha F[E_s[V(\lambda y.\#_\alpha(\lambda u.\#_\beta F'[u\bullet])(\lambda z.E_s[y\bullet]))]] \\
&\to \#_\alpha F[E_s[V(\lambda y.\#_\alpha \#_\beta F'[E_s[y\bullet]])]] \\
&\to \#_\alpha F[E_s[V(\lambda y.\#_\alpha F[E_s[y\bullet]])]] \\
&\equiv \#_\alpha E_0[V(\lambda u.\#_\alpha E_0[u\bullet])]
\end{aligned}$$

where F' is the result of substituting β for free occurrences of α in F to avoid conflict of bound variables.

Case-3-3. If $E_0 \equiv F[\#_\beta[\ ]]$ where F is an evaluation context, then we have (assuming V is $\lambda x.M$)

$$\begin{aligned}
E[\#_\alpha F[\#_\beta \mathtt{calldc}_\alpha \lambda x.M]] &\to_1 E[\#_\alpha F[\mathtt{calldc}_\alpha \lambda x.\#_\beta M]] \\
&= E[\#_\alpha F[(\lambda x.\#_\beta M)(\lambda u.\#_\alpha F[u\bullet])]] \\
&\qquad \text{(by induction hypothesis)} \\
&\to E[\#_\alpha F[\#_\beta M[\lambda u.\#_\alpha F[u\bullet]/x]]]
\end{aligned}$$

On the other hand, we have

$$\begin{aligned} E[\#_\alpha F[\#_\beta(\lambda x.M)(\lambda u.\#_\alpha F[\#_\beta u\bullet])]] &\to_1 E[\#_\alpha F[\#_\beta(\lambda x.M)(\lambda u.\#_\alpha F[u\bullet])]] \\ &\to_1 E[\#_\alpha F[\#_\beta M[\lambda u.\#_\alpha F[u\bullet]/x]]] \end{aligned}$$

So we have the equality. When V is not in the form $\lambda x.M$, namely, it is a variable or a constant, then the proof is easier since there are no free occurrences of the tag β.

Case-4. $E[\#_\alpha E_0[\mathtt{throw}_\alpha V]] \rightsquigarrow E[V]$

Since $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ does not allow λ-abstraction for the above V, V must be either a variable or a constant. We then prove this case similarly to the Case-3. If E_0 is composed by a delimiter, namely, $E_0 \equiv \#_\beta[\ ]$, then we use the fact that V does not contain β free. We also have $E[\#_\alpha E'[\mathtt{throw}_\alpha V]] \to E[\#_\alpha V]$, and $\#_\alpha V \to_1 V$. Other cases are easy. □

By this theorem and the confluence of $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$, we have the following corollary, which means our reduction rules really reflect the intended operational semantics.

Corollary 1 (Correspondence of $\rightsquigarrow$ and $\to$ in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$). *Let M be a closed term and V be a value. If $M \rightsquigarrow V$, then $M \to V$ in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$.*

7 A Logical View

The Curry-Howard isomorphism relates typed lambda calculi and intuitionistic logical systems. As Griffin and other researchers showed, the isomorphism can be extended to the relationship between typed lambda calculi with sequential control operators and classical logic [10]. In this section, we show that $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$ also correspond to classical logic.

$\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$ are Classical Logic

We assume that $\bot$ is included as an atomic type in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$. (If it is not the case, choose any atomic type as $\bot$, since we do not use the $\bot$-elimination rule.) Let ϕ be a map which simply discards the lefthand-side of colon from a judgement $M : A$, namely, $\phi(M : A) \equiv A$. A type in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ and $\lambda_{\mathtt{DC}}$ can be regarded as a formula in implicational logic where $\to$ is interpreted by implication, $\neg A$ is interpreted by $A \supset \bot$, and $\mathtt{Unit}$ is a provable formula (say, $\bot \to \bot$). The map ϕ naturally extends to type environments.

Theorem 6 (Isomorphism between $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}/\lambda_{\mathtt{DC}}$ and classical logic). *Let Γ be a type environment, M be a preterm, and A be a type (a formula). Then $\Gamma \vdash M : A$ holds in $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ if and only if $\phi(\Gamma) \vdash A$ holds in a classical implicational logic. The same thing holds for $\lambda_{\mathtt{DC}}$.*

Proof (only-if part). We only have to verify the following three typing rules are preserved through ϕ.

(i) For the case of the delimiter, we have to infer $\Gamma \vdash B$ from $\Gamma \cup \{\neg B\} \vdash B$. (ii) For the case of `calldc`, we have to infer $\Gamma \cup \{\neg B\} \vdash A$ from $\Gamma \vdash ((\texttt{Unit} \to A) \to B) \to A$ ($\texttt{Unit}$ is some provable formula). (iii) For the case of `throw`, we have to infer $\Gamma \cup \{\neg B\} \vdash C$ from $\Gamma \vdash B$.

All of them can be proved in classical logic.

Proof (if part).

We only have to show the classical reasoning can be simulated by $\lambda_{\texttt{DC}}^{\text{atomic}}$. To show this, we shall prove that, for any type A and B, there exists a closed term of type $((A \to B) \to A) \to A$. This is shown by induction on the type A.

For the base case (A is an atomic type K), the following figure shows that $((K \to B) \to K) \to K$ is inhabited.

$$\dfrac{\dfrac{\{y : (K \to B) \to K\} \vdash y : (K \to B) \to K \qquad \dfrac{\dfrac{\overline{\{x : K\} \vdash x : K}}{\{x : K, \alpha : \neg K\} \vdash \texttt{throw}_\alpha x : B}}{\{\alpha : \neg K\} \vdash \lambda x.\texttt{throw}_\alpha x : K \to B}}{\{y : (K \to B) \to K, \alpha : \neg(K)\} \vdash y(\lambda x.\texttt{throw}_\alpha x) : K}}{\dfrac{\{y : (K \to B) \to K\} \vdash \#_\alpha y(\lambda x.\texttt{throw}_\alpha x) : K}{\{\} \vdash \lambda y.\#_\alpha y(\lambda x.\texttt{throw}_\alpha x) : ((K \to B) \to K) \to K}}$$

For the induction step, let A be $C \to D$. By induction hypothesis, we have a closed term M of type $((D \to B) \to D) \to D$. Then we can easily show that a closed term $\lambda ux.M(\lambda y.u(\lambda z.y(zx))x)$ has type $((A \to B) \to A) \to A$. Hence we can prove all the instances of Peirce's law in $\lambda_{\texttt{DC}}^{\text{atomic}}$.

The proof is even easier for $\lambda_{\texttt{DC}}$, since we can use an arbitrary type in place of K in the figure above. □

Note that we used the `throw`-operator only, which means that $\lambda_{\texttt{DC}}^{\text{atomic}}$ (and $\lambda_{\texttt{DC}}$) without the `calldc`-operator is still a classical calculus. In fact, Sato and the present author developed calculi for the catch/throw mechanism [18,14] which correspond to classical propositional logic. It is easy to see that the delimiter and the `throw`-operator can be understood as the catch and the throw operators, and such a subcalculus of $\lambda_{\texttt{DC}}^{\text{atomic}}$ (or $\lambda_{\texttt{DC}}$) constitutes a confluent subcalculus of the classical catch/throw calculi in [18,14].

Note also that our reduction rules can be thought as proof reductions in classical logic. We already gave the subject reduction for the reduction (6), and if we forget the terms, we obtain a proof reduction. The resulting proof-reduction is (by nature) similar to Griffin's reduction rules for Felleisen's control operator.

The `calldc`-operator is not Classical

On the other hand, if we eliminate the `throw`-operator from $\lambda_{\texttt{DC}}^{\text{atomic}}$ (or $\lambda_{\texttt{DC}}$), the resulting calculus becomes strictly weaker in the logical sense.

Suppose $\{\} \vdash M : A$ is inferred in $\lambda_{\texttt{DC}}$ where M does not contain the `throw`-operator. We shall show that A is provable in minimal logic. To simplify the

matter, we assume M uses only one tag variable α. Suppose M contains k subterms of the form $\mathtt{calldc}_\alpha N_i$, and the type of N_i is $((\mathtt{Unit} \to A_i) \to B) \to A_i$. We put $P_i \equiv (((\mathtt{Unit} \to A_i) \to B) \to A_i) \to A_i$. The introduction rule of `calldc` is (when mapped by ϕ) provable if we assume each P_i. Hence we can regard the delimiter introduction rule (through ϕ) as eliminating $P_1, \cdots, P_k$ from the assumption list. In other words, our goal is to prove $\Gamma \vdash B$ from $\Gamma, P_1, \cdots, P_k \vdash B$. But the formula $(\bigwedge_{i=1}^{k} P_i \to B) \to B$ is provable in minimal logic (which can be shown by induction on k).

From this fact, one may think that the `calldc`-operator may be expressible by standard combinators such as S and K, but we believe this is not the case. The proof term of the above theorem has the same type as our control operators, but it behaves quite differently (the latter term is not interesting in computational aspects).

8 Conclusion

Partial continuations were proposed by Felleisen and others and there are many researches on partial continuations since then. Compared to existing calculi for partial continuations, the characteristic feature of our approach is that our calculus is based on a type-theoretic framework. We showed that our calculus (i) enjoys the subject reduction property (ii) is confluent, (iii) does admit the standard Curry-Howard isomorphism (by which it corresponds to classical logic). Hieb et al's subcontinuation also has static scope, but their approach also lacks the type-safety property (which means that it sometimes generates uncaught partial-continuation object). Our approach can be thought as a refinement of (a typed version of) Hieb et al's subcontinuations, and we believe that our calculus can be a basis of syntactic, type-theoretic analysis for partial continuations and other variations of control operators.

Since we can abstract tags, we believe that many examples by Felleisen's one and Danvy and Filinski's one can be written in our calculus. In fact we already worked in tree-traversal example by Felleisen [6].

Future Work. So far, several research topics are left for future work. The first target is the strong normalization (SN) of $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$. A common tool to show it is a type-preserving CPS-translation. We gave a CPS-translation for $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ in our draft, but it does not preserve typing, so the SN of $\lambda_{\mathtt{DC}}^{\mathrm{atomic}}$ is an open problem. Other directions are expressiveness and application. In this paper, we confined ourselves to sequential programs, but as many authors pointed out, partial continuations are a useful tool for giving control over parallel/concurrent programs. Also, there should be applications for formalizing mobile computing.

Acknowledgement

The author would like to thank Olivier Danvy, Kenichi Asai and anonymous referees for pointing out errors in the earlier drafts and let him know references. He would also like to thank Masahiko Sato and Izumi Takeuti for helpful discussions. This work was supported in part by Grant-in-Aid for Scientific Research from the Ministry of Education, Science and Culture of Japan, No. 11780213.

References

1. Kelsey, R., W. Clinger, and J. Rees (eds.): Revised^5 Report on the Algorithmic Language Scheme, 1998.
2. Danvy, O. and A. Filinski: Abstracting Control, Proc. 1990 ACM Conference on Lisp and Functional Programming, pp. 151-160, 1990.
3. Danvy, O. and A. Filinski: Representing Control: a Study of the CPS Transformation, Mathematical Structures in Computer Science 2(4), pp. 361-391, 1992.
4. de Groote, P.: A Simple Calculus of Exception Handling, Typed Lambda Calculi and its Applications (Dezani-Ciancaglini, M. and G. Plotkin eds.), Lecture Notes in Computer Science **902**, pp. 201–215. Springer, 1995.
5. Felleisen, M., D. Friedman, E. Kohlbecker, and B. Duba: A Syntactic Theory of Sequential Control, Theoretical Computer Science 52, pp. 205-237, 1987.
6. Felleisen, M.: The Theory and Practice of First-Class Prompts, Proc. 15th ACM Symp. on Principles of Programming Languages, pp. 180-190, 1988.
7. Felleisen, M., M. Wand, D. Friedman, and B. Duba: Abstract Continuations: A Mathematical Semantics for Handling Full Functional Jumps, Proc. 1988 ACM Conf. on Lisp and Functional Programming, pp. 52-62, 1988.
8. Felleisen, M., and R. Hieb: The Revised Report on the Syntactic Theories of Sequential Control and State, Theoretical Computer Science 103, pp. 235-271, 1992.
9. Filinski, A.: Representing Monads, Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 446-457, 1994.
10. Griffin, T.: A Formulae-as-Types Notion of Control, Conference Record of 17th ACM Symp. on Principles of Programming Languages, pp. 47-58, 1990.
11. Gunter, C. A., D. Remy, and J. G. Riecke: A Generalization of Exceptions and Control in ML-Like Languages, Functional Programming and Computer Architecture, pp. 12-23, 1995.
12. Harper, R., B. Duba, and D. Macqueen: Typing First-Class Continuations in ML, J. Functional Programming 3(4), pp. 465-484, 1993.
13. Hieb, R., R. Dybvig, and C. W. Anderson: Subcontinuations, Lisp and Symbolic Computation 6, pp. 453–484, 1993.
14. Kameyama, Y. and M. Sato: Strong Normalizability of the Non-deterministic Catch/Throw Calculii, Theoretical Computer Science, to appear.
15. Kameyama, Y.: A Type System for Delimited Continuations (Preliminary Version), Workshop on Programming and Programming Languages (PPL2000), Japan Society for Software Science and Technology, Kanzanji, Japan, Mar. 2000.
16. Ong, C.-H. L. and C. A. Stewart: A Curry-Howard Foundation for Functional Computation, Proc. 24th ACM Symposium on Principles of Programming Languages, pp. 215–227, 1997.
17. Parigot, M.: $\lambda\mu$-calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science **624**, Springer, pp. 190-201, 1992.
18. Sato, M.: Intuitionistic and Classical Natural Deduction Systems with the Catch and the Throw Rules, Theoretical Computer Science Vol. 175, No. 1, pp. 75-92, 1997.
19. Takahashi, M: Parallel Reductions in λ-Calculus, J. Symbolic Computation **7**, 1989, pp. 113–123.
20. Queinnec, C. and B. Serpette: A Dynamic Extent Control Operator for Partial Continuation, Proc. 18th ACM Symp. on Principles of Programming Languages, pp. 174-184, 1991.

Partially Typed Terms between Church-Style and Curry-Style

Ken-Etsu Fujita[1] and Aleksy Schubert[2]

[1] Shimane University, Matue 690-8504, Japan,
fujiken@cis.shimane-u.ac.jp
[2] Warsaw University, 02-097 Warsaw, Poland,
alx@mimuw.edu.pl

Abstract. We introduce several structures between Church-style and Curry-style based on partially typed terms formalism. In the uniform framework, we study the static properties of the λ-terms between the two styles. It is proved that type checking, typability, and type inference for domain-free $\lambda 2$ are in general undecidable. A simple instance of the second-order unification problem is reduced to the problem of type inference for domain-free $\lambda 2$. The typability problem is undecidable even for a predicative fragment of domain-free $\lambda 2$, called the rank 2 fragment. It is also found that making polymorphic domains free and the use of type-holes [ ] are independently responsible for the undecidability of the partial polymorphic type reconstruction problem.

1 Introduction

There are known three styles of (typed) λ-terms, called Curry-style, Church-style, and domain-free style. For some systems such as simply typed λ-calculus and ML [21,9], it is well-known that the Curry-style and the corresponding Church-style are essentially equivalent [1,15]. Hence, the Curry system serves as a short-hand for the Church system. On the other hand, recently, Barthe, Sørensen, and Hatcliff [3,4] introduced the notion of domain-free pure type system. Terms in domain-free style have domain-free λ-abstraction. Barthe and Sørensen posed a question to know whether the problem of type checking is decidable for domain-free $\lambda 2$ and $\lambda\omega$ (page 18 [4]). In this paper, we will show that type checking, typability, and type inference are, in general, undecidable for domain-free $\lambda 2$. In order to prove this, we reduce simple instances of the second-order unification problem to the problem of strong type inference for domain-free $\lambda 2$.

Original motivation for domain-free systems comes from a study of classical type system which is an extension of intuitionistic type theory together with classical rules such as double negation elimination. The domain-free systems are useful to give continuation-passing style translations [3,10] which provide a certain semantics of classical type system. Further, when we construct a polymorphic call-by-value calculus with control operators such as `callcc` or μ-operators [24], the Curry style cannot work for a consistent system. For instance, see the traditional counterexample (ML with `callcc` is unsound) by Harper&Lillibridge

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 505–520, 2000.
© Springer-Verlag Berlin Heidelberg 2000

[14], and see also a proof-theoretical observation in [10]. Hence, domain-free $\lambda\mu$-calculus has been introduced in [10], where the explicit type annotations for polymorphic terms play a role of choosing an appropriate computation under call-by-value. Our result in this paper also gives a negative answer to the problem of type checking for second-order $\lambda\mu$-calculus in domain-free style, which is a variant of Parigot's $\lambda\mu$-calculus in Curry style [24].

Domain-free systems are also useful for a study of partial polymorphic type reconstruction. Boehm [2] and Pfenning [25] have proven that the partial type reconstruction problem is, in general, undecidable for second-order λ-calculus. The typability problem for domain-free $\lambda 2$ can be regarded as a special case of the problem of type reconstruction for partially typed terms. Our result in this paper means that the restricted problem of type reconstruction for partially typed terms is still undecidable. Moreover, observation of the undecidability proof reveals that the typability problem is undecidable even for a predicative fragment of domain-free $\lambda 2$, called the rank 2 fragment [20,17]. This analysis also implies the involved result that the partial type reconstruction problem is still undecidable for the rank 2 fragment of second-order λ-calculus, contrary to the decidable typability for the rank 2 fragment of $\lambda 2$ in Curry style [17]. From the viewpoint of partially typed terms, we introduce fine structures between Church-style and Curry-style, including the domain-free style. In the uniform framework, we study the static properties of the λ-terms between the two styles. It is found that making polymorphic domains free and the use of type-holes [ ] are independently responsible for the undecidability of the partial type reconstruction problem. In this sense, this work can give a guide to the construction of typed languages with decidable type checking and typability.

2 Curry-Style, Church-Style, and Domain-Free

In Curry-style, terms are essentially type free [7,8,16], and types can be assigned by rules of a type theory if well-formed. Terms in the Church-style typed λ-calculus, on one hand, are originally defined only from variables uniquely type annotated [6]. Following Curry's philosophy, today one has the notion of pseudo-terms [1] separated from a type theory. On the other hand, terms in domain-free (DF) style have domain-free λ-abstraction [4], and second-order λ-calculus in domain-free style can be regarded, in a sense, as an intermediate representation between à la Curry and à la Church, as shown in the following table:
Types $\sigma ::= t \mid \sigma \to \sigma \mid \forall t.\sigma$

Styles of (typed) $\lambda 2$-terms

$\lambda 2$-(pseudo)terms	object-var abst.	term app.	type-var abst.	type app.
Church-style	$\lambda x{:}\sigma.M$	MM	$\lambda t.M$	$M[\sigma]$
Domain-Free	$\lambda x.N$	NN	$\lambda t.N$	$N[\sigma]$
Curry-style	$\lambda x.U$	UU		

We give a definition of domain-free $\lambda 2$-calculus. In terms of domain-free pure type systems [3,4], this domain-free system is constructed from sorted variables; a metavariable for variables of the first sort (term variables) is x and a metavariable for variables of the second sort (type variables) is t. Then, on the basis of the sorted variables, type abstraction can be represented by λt rather than the traditional Λt, and we also have explicit distinction between terms and types.

Type Assignment Rules for Domain-Free $\lambda 2$

$$\Gamma \vdash x : \Gamma(x)$$

$$\frac{\Gamma \vdash N_1 : \sigma_1 \to \sigma_2 \quad \Gamma \vdash N_2 : \sigma_1}{\Gamma \vdash N_1 N_2 : \sigma_2}\ (\to E) \qquad \frac{\Gamma, x{:}\sigma_1 \vdash N : \sigma_2}{\Gamma \vdash \lambda x.N : \sigma_1 \to \sigma_2}\ (\to I)$$

$$\frac{\Gamma \vdash N : \forall t.\sigma_1}{\Gamma \vdash N[\sigma_2] : \sigma_1[t := \sigma_2]}\ (\forall E) \qquad \frac{\Gamma \vdash N : \sigma}{\Gamma \vdash \lambda t.N : \forall t.\sigma}\ (\forall I)^*$$

where $(\forall I)^*$ denotes the eigenvariable condition.

The introduction rules, $(\to I)$ and $(\forall I)$ can be coded, respectively, as domain-free λ-abstractions based on the distinction between the sorted variables. The elimination rules, $(\to E)$ and $(\forall E)$ can also be represented, respectively, by the pairs of two expressions, based on sorted variables. Hence, when well-typed terms of domain-free $\lambda 2$ are given, the type assignment rules are uniquely determined by the shape of the terms. From this syntactical property of terms, we have the natural generation lemma for domain-free $\lambda 2$.

Lemma 1 (Generation lemma) (1) *If* $\Gamma \vdash x : \sigma$, *then* $\Gamma(x) = \sigma$.
(2) *If* $\Gamma \vdash N_1 N_2 : \sigma$, *then* $\Gamma \vdash N_1 : \sigma_1 \to \sigma$ *and* $\Gamma \vdash N_2 : \sigma_1$ *for some* σ_1.
(3) *If* $\Gamma \vdash \lambda x.N : \sigma$, *then* $\Gamma, x : \sigma_1 \vdash N : \sigma_2$ *and* $\sigma \equiv \sigma_1 \to \sigma_2$ *for some* σ_1 *and* σ_2.
(4) *If* $\Gamma \vdash \lambda t.N : \sigma$, *then* $\Gamma \vdash N : \sigma_1$ *and* $\sigma \equiv \forall t.\sigma_1$ *together with* $t \notin FV(\Gamma)$ *for some* σ_1.
(5) *If* $\Gamma \vdash N[\sigma_1] : \sigma$, *then* $\Gamma \vdash N : \forall t.\sigma_2$ *and* $\sigma \equiv \sigma_2[t := \sigma_1]$ *for some* σ_2.

3 Type Checking, Typability, and Type Inference for Domain-Free $\lambda 2$

The problem of type inference is a problem; given a term M, are there a context Γ and a type σ such that $\Gamma \vdash M : \sigma$ is derivable? On one hand, the problem of strong type inference [30] is a problem; given a term M and a context Γ_0, are there a context $\Gamma \supseteq \Gamma_0$ and a type σ such that $\Gamma \vdash M : \sigma$ is derivable? The typability problem is a problem; given a term M and a context Γ, is there a type σ such that $\Gamma \vdash M : \sigma$ is derivable? Finally, the type checking problem is a problem; given a term M, a type σ, and a context Γ, is the judgement $\Gamma \vdash M : \sigma$ derivable? The problems of (strong) type inference, typability, and type checking are denoted, respectively, by $? \vdash M :?$, $\Gamma \vdash M :?$, and $\Gamma \vdash M : \sigma?$.

The problems of type checking, typability, and type inference for Curry and Church $\lambda 2$ are investigated by Jutting [31], Wells [32], and Schubert [28,29], as shown in the following table:

Decidability for type checking, typability, and type inference of $\lambda 2$

$\lambda 2$	$\Gamma \vdash M : \sigma ?$	$\Gamma \vdash M :?$	$? \vdash M :?$
Church-style	yes [31]	yes [31]	no [28]
Domain-Free	$?_1$	$?_2$	$?_3$
Curry-style	no [32]	no [32]	no [32]

In this paper, we will prove that in the table above, all of $?_1$, $?_2$, and $?_3$ (case of strong type inference) are "no", i.e., undecidable[1].

By the use of a type forgetful map, the three styles of judgements are equivalent in the following sense, where $|\ |$ is a domain erasing map ($|\lambda x : \sigma.M| = \lambda x.|M|$), and $||\ ||$ is a type erasing map ($||M\sigma|| = ||M||$, $||\lambda t.M|| = ||M||$):
(1) If $\Gamma \vdash M : \sigma$ in Church style, then $\Gamma \vdash |M| : \sigma$ in domain-free style.
(2) If $\Gamma \vdash M : \sigma$ in domain-free style, then $\Gamma \vdash ||M|| : \sigma$ in Curry style.

The inverse directions say that there exists a term whose erasure is the same as the original term [15].
(-1) If $\Gamma \vdash M : \sigma$ in domain-free style, then there exists M_1 such that $\Gamma \vdash M_1 : \sigma$ in Church style and $|M_1| \equiv M$.
(-2) If $\Gamma \vdash M : \sigma$ in Curry style, then there exists M_2 such that $\Gamma \vdash M_2 : \sigma$ in domain-free style and $||M_2|| \equiv M$.

For the problems above, however, the inverse directions are not straightforward, since the forgetful maps are not one-to-one. We have to check that the given term is the same as an erasure of *some* term, or that the given term has the same type as that of the erasure, see also Section 7. Here, we will directly study the problems for domain-free $\lambda 2$.

On the basis of the generation lemma (Lemma 1), we first observe that the (strong) type inference problem for domain-free $\lambda 2$ is reduced to the typability problem, and the typability problem for domain-free $\lambda 2$ is reduced to the type checking problem.

Lemma 2 $\exists \Gamma . \exists \sigma .\ \Gamma, \Gamma_0 \vdash M : \sigma$ *in* DF $\lambda 2 \iff \exists \sigma .\ \Gamma_0 \vdash \lambda \overrightarrow{x}.M : \sigma$ *in* DF $\lambda 2 \iff \Gamma_0 \vdash (\lambda x.\lambda y.y)(\lambda \overrightarrow{x}.M) : t \to t$ *in* DF $\lambda 2$

4 Terms with Partial Type Annotations

In order to study static properties of λ-terms in a uniform framework, we introduce partially typed terms (preterms) and type assignment rules for preterms.

Partially typed terms (preterms), denoted by P, Q, are:
$P ::= x \mid \lambda x{:}\sigma.P \mid PP \mid \lambda t.P \mid P[\sigma]^a \mid \lambda x.P \mid P[\ ]^a$

[1] After completing [11,12], a correspondence with Gilles Barthe informed the author that all of ? are independently proved undecidable in [5]. For the detail, see also the footnote in Section 7.

where the mark $[\,]^a$ must be left to indicate that a type has been erased. Moreover, the label a in $[\,]^a$ will be used to identify the occurrences of $[\,]$, i.e., the type-holes $[\,]$ with the same label should be obtained by erasing the same type. This annotation is a natural constraint to the traditional definition of preterms [2,25] and gives a useful information. The use of our type-holes $[\,]^a$ plays the same role as existential quantification over types, see also Proposition 4. We often omit the annotation of a type-hole unless it is necessary to identify the occurrences.

We say that a preterm P is a partial erasure of M in Church-style, denoted by $P \preceq_1 M; \Delta$ if it is derived by the following rules, where Δ is a mapping from an annotation to a type:

$$\frac{}{x \preceq_1 x;}\ (\text{var})$$

$$\frac{P \preceq_1 M; \Delta}{\lambda x{:}\sigma.P \preceq_1 \lambda x{:}\sigma.M; \Delta}\ (\text{abst–var}) \qquad \frac{P \preceq_1 M; \Delta}{\lambda x.P \preceq_1 \lambda x{:}\sigma.M; \Delta}\ (\text{abst–df})$$

$$\frac{P_1 \preceq_1 M_1; \Delta \quad P_2 \preceq_1 M_2; \Delta}{P_1P_2 \preceq_1 M_1M_2; \Delta}\ (\text{app}) \qquad \frac{P \preceq_1 M; \Delta}{\lambda t.P \preceq_1 \lambda t.M; \Delta}\ (\text{abst–type})$$

$$\frac{P \preceq_1 M; \Delta}{P[\sigma]^a \preceq_1 M[\sigma]^a; \Delta, \sigma^a}\ (\text{app–type})^\sharp \qquad \frac{P \preceq_1 M; \Delta}{P[\,]^a \preceq_1 M[\sigma]^a; \Delta, \sigma^a}\ (\text{app–hole})^\sharp$$

Here, $\sharp$ denotes the condition such that if $\Delta(a) \neq \emptyset$ then $\Delta(a) = \sigma$.

We now consider special cases of preterms, which give fine structures between Church-style and Curry-style, as follows:
(1) Domain-free terms:
Preterms which are derived by the use of all the rules but (abst-var) and (app-hole).
(2) $[\,]$-application terms:
Preterms which are derived by all the rules but (abst-df) and (app-type).
(3) DF&$[\,]$ terms:
Preterms which are derived by all the rules but (abst-var) and (app-type).
A term in Curry-style is here regarded as a full erasure that is obtained from a term of DF&$[\,]$ by deleting both λt and $[\,]$.

We also say that a preterm P_1 is a partial erasure of a preterm P_2, denoted by $P_1 \preceq P_2$ if it is derived by the following rules:

$$x \preceq x \qquad \frac{P_1 \preceq P_2}{\lambda x{:}\sigma.P_1 \preceq \lambda x{:}\sigma.P_2}$$

$$\frac{P_1 \preceq P_2}{\lambda x.P_1 \preceq \lambda x{:}\sigma.P_2} \qquad \frac{P_1 \preceq P_2}{\lambda x.P_1 \preceq \lambda x.P_2} \qquad \frac{P_1 \preceq Q_1 \quad P_2 \preceq Q_2}{P_1P_2 \preceq Q_1Q_2}$$

$$\frac{P_1 \preceq P_2}{\lambda t.P_1 \preceq \lambda t.P_2} \qquad \frac{P_1 \preceq P_2}{P_1[\sigma] \preceq P_2[\sigma]} \qquad \frac{P_1 \preceq P_2}{P_1[\,] \preceq P_2[\sigma]} \qquad \frac{P_1 \preceq P_2}{P_1[\,] \preceq P_2[\,]}$$

It is clear that $\preceq$ constitutes a partial order on preterms.

We next define type assignment rules for preterms:

$$\Gamma \vdash_p x : \Gamma(x); \Delta$$

$$\frac{\Gamma \vdash_p P_1 : \sigma_1 \to \sigma_2; \Delta \quad \Gamma \vdash_p P_2 : \sigma_1; \Delta}{\Gamma \vdash_p P_1 P_2 : \sigma_2; \Delta}\ (\to E)$$

$$\frac{\Gamma, x{:}\sigma_1 \vdash_p P : \sigma_2; \Delta}{\Gamma \vdash_p \lambda x{:}\sigma_1.P : \sigma_1 \to \sigma_2; \Delta}\ (\to I_1) \qquad \frac{\Gamma, x{:}\sigma_1 \vdash_p P : \sigma_2; \Delta}{\Gamma \vdash_p \lambda x.P : \sigma_1 \to \sigma_2; \Delta}\ (\to I_2)$$

$$\frac{\Gamma \vdash_p P : \forall t.\sigma_1; \Delta}{\Gamma \vdash_p P[\sigma_2]^a : \sigma_1[t := \sigma_2]; \Delta}\ (\forall E_1)^{\sharp\sharp} \qquad \frac{\Gamma \vdash_p P : \forall t.\sigma_1; \Delta}{\Gamma \vdash_p P[\,]^a : \sigma_1[t := \sigma_2]; \Delta}\ (\forall E_2)^{\sharp\sharp}$$

$$\frac{\Gamma \vdash_p P : \sigma; \Delta}{\Gamma \vdash_p \lambda t.P : \forall t.\sigma; \Delta}\ (\forall I)^*$$

where $(\forall I)^*$ denotes the eigenvariable condition, and $\sharp\sharp$ does $\Delta(a) = \sigma_2$.

We may write $\Gamma \vdash_p P : \sigma$ for $\Gamma \vdash_p P : \sigma; \Delta$ and $[\,]$ for $[\,]^a$, unless it is necessary to use annotations for type-holes. Since a well-typed preterm can be uniquely decomposed by one of the assignment rules, we have the generation lemma below:

Lemma 3 (Generation lemma) (1) *If $\Gamma \vdash_p x : \sigma; \Delta$, then $\Gamma(x) = \sigma$.*
(2) *If $\Gamma \vdash_p P_1 P_2 : \sigma; \Delta$, then $\Gamma \vdash_p P_1 : \sigma_1 \to \sigma; \Delta$ and $\Gamma \vdash P_2 : \sigma_1; \Delta$ for some σ_1.*
(3) *If $\Gamma \vdash_p \lambda x : \sigma_1.P : \sigma; \Delta$, then $\Gamma, x : \sigma_1 \vdash_p P : \sigma_2; \Delta$ and $\sigma \equiv \sigma_1 \to \sigma_2$ for some σ_2.*
(4) *If $\Gamma \vdash_p \lambda x.P : \sigma; \Delta$, then $\Gamma, x : \sigma_1 \vdash_p P : \sigma_2; \Delta$ and $\sigma \equiv \sigma_1 \to \sigma_2$ for some σ_1 and σ_2.*
(5) *If $\Gamma \vdash_p \lambda t.P : \sigma; \Delta$, then $\Gamma \vdash_p P : \sigma_1; \Delta$ together with $\sigma \equiv \forall t.\sigma_1$ and $t \notin FV(\Gamma)$ for some σ_1.*
(6) *If $\Gamma \vdash_p P[\sigma_1]^a : \sigma; \Delta$, then $\Gamma \vdash P : \forall t.\sigma_2; \Delta$ together with $\sigma \equiv \sigma_2[t := \sigma_1]$ and $\Delta(a) = \sigma_1$ for some σ_2.*
(7) *If $\Gamma \vdash_p P[\,]^a : \sigma; \Delta$, then $\Gamma \vdash P : \forall t.\sigma_2; \Delta$ together with $\sigma \equiv \sigma_2[t := \sigma_1]$ and $\Delta(a) = \sigma_1$ for some σ_2.*

The above mentioned generation lemma allows to deduce the following relationship between Church-style judgements and judgements of preterms.

Lemma 4 (1) *If we have $\Gamma \vdash_p P_1 : \sigma$, then $\Gamma \vdash_p P_2 : \sigma$ for any $P_2 \preceq P_1$.*
(2) *If we have $\Gamma \vdash_p P : \sigma; \Delta$, then $\Gamma \vdash M : \sigma$ in Church style for some $M \succeq_1 P; \Delta$.*

The problem of partial type reconstruction [25,26] is a problem; given a context Γ and a preterm P, is there a Church-style term M such that $\Gamma \vdash M : \sigma$ holds in Church-style for some type σ and that $P \preceq_1 M; \Delta$ for some Δ? From the lemma 4, the partial type reconstruction problem is equivalent to the following typability problem:
Given a context Γ and a preterm P, is there a type σ such that $\Gamma \vdash_p P : \sigma; \Delta$ is derivable for some Δ?

We say that a preterm P is a normal form if P contains a subterm in the form of neither $(\lambda x{:}\sigma.P_1)P_2$, $(\lambda x.P_1)P_2$, $(\lambda t.P_1)\sigma$, nor $(\lambda t.P_1)[\,]$.

5 Type Inference is Undecidable for Domain-Free $\lambda 2$

In this section, we prove that the problem of strong type inference for domain-free $\lambda 2$ is undecidable. To show this, we demonstrate a stronger result such that the problem of strong type inference is undecidable for the predicative fragment of domain-free $\lambda 2$, called domain-free ML. Strictly speaking, the following system is a subsystem of the so-called ML, however such a subsystem is enough to establish the undecidability.

Domain-free (DF) ML:
Monotypes $\tau ::= t \mid \tau \to \tau$ Polytypes $\sigma ::= \tau \mid \forall t.\sigma$
Contexts $\Gamma ::= \langle\ \rangle \mid x{:}\sigma, \Gamma$
Terms $M ::= x \mid \lambda x.M \mid MM \mid x[\tau_1]\cdots[\tau_n]$
Type Assignment Rules

$$\frac{\Gamma(x) = \forall t_1 \cdots t_n.\tau}{\Gamma \vdash x[\tau_1]\cdots[\tau_n] : \tau[t_1 := \tau_1, \cdots t_n := \tau_n]}\ (n \geq 0)$$

$$\frac{\Gamma, x{:}\tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x.M : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2}$$

Remarked that [ ]-application ML can also be defined similarly.

We first introduce a restricted second-order unification problem for well-formed second-order expressions, which are defined from monotypes τ, binary function constant $\to$, and n-arity second-order function variable X ($n \geq 0$) whose arguments contain no variables. Such terms for the second-order unification are denoted by T or U. A well-formed expression is defined as follows:
(1) A type variable t is a well-formed expression.
(2) If X is an n-arity variable ($n \geq 0$) and τ_i ($1 \leq i \leq n$) are monotypes, then $X\tau_1 \cdots \tau_n$ is well-formed.
(3) If T_1 and T_2 are well-formed, then so is $T_1 \to T_2$.

Given a well-formed expression T, a set of (unification) variables in T denoted by $uVar(T)$ and a set of constants in T denoted by $Con(T)$ are defined, respectively, as follows:
$uVar(t) = \emptyset$; $uVar(X\tau_1 \cdots \tau_n) = \{X\}$ ($n \geq 0$);
$uVar(T_1 \to T_2) = uVar(T_1) \cup uVar(T_2)$.
$Con(t) = \{t\}$; $Con(X\tau_1 \cdots \tau_n) = \emptyset$ ($n \geq 0$);
$Con(T_1 \to T_2) = Con(T_1) \cup Con(T_2)$.

Given well-formed T_1 and T_2. Let $uVar(T_1, T_2)$ be $uVar(T_1) \cup uVar(T_2) = \{X_1, \cdots, X_n\}$. The unification problem $(T_1 \doteq T_2)$ is a problem to find well-formed U_i for each X_i ($1 \leq i \leq n$), such that
(1) Let X_i be $k(i)$-arity variable, and S be a substitution such that $[X_1 := \lambda t_1 \cdots t_{k(1)}.U_1, \cdots, X_n := \lambda t_1 \cdots t_{k(n)}.U_n]$. Then $S(T_1) =_\beta S(T_2)$ holds.
(2) We have $uVar(U_i) = \emptyset$ for $1 \leq i \leq n$.

If we have a substitution S such that the above (1) and (2) are satisfied, then we say that T_1 and T_2 are unifiable, and that the unification problem has an

answer S. In this case, from the definition, there exists a monotype τ such that $S(T_1) =_\beta \tau =_\beta S(T_2)$.

Theorem 1 (Schubert[28]). *The second-order unification problem on the well-formed expressions is undecidable.*

Schubert [28] has proved that the halting problem for two-counter automata is reduced to the unification problem, where a two-counter automaton can simulate an arbitrary Turing machine.

In order to give a reduction from the unification problem to the problem of type inference for domain-free ML, we first define a (pre)context Σ. The context itself may not be an ML-context, but it becomes an ML-context under some substitution if unifiable. This can be justified, since the reduction is formalized as follows: the unification problem $T_1 \doteq T_2$ has an answer if and only if there exist Γ and τ such that $\Gamma, \Gamma_0 \vdash M : \tau$ in domain-free ML. Here, Γ_0 and M are given by T_1 and T_2, where Γ_0 consists only of monotypes in $Con(T_1, T_2)$. Only if T_1 and T_2 are unifiable, say the unifier S, then the ML-context Γ can be obtained as a subcontext of $S(\Sigma)$, such that $S(\Sigma) = \Gamma, \Gamma_0$. Moreover, the monotype τ can also be obtained as a substitution instance (of $ty(\cdot)$ defined below) by S.

Given a well-formed T, then we construct the context $\Sigma[T]$, such that
(1) For each $t \in Con(T)$, t is inhabited in $\Sigma[T]$, i.e., $\Sigma[T](x) = t$ for some x.
(2) For each n-arity variable $X \in uVar(T)$ where $n \geq 0$, the universal closure $\forall t_1 \cdots t_n.(Xt_1 \cdots t_n)$ is inhabited in $\Sigma[T]$.

$\Sigma[T_1, T_2]$ is also defined similarly, and we simply write Σ for $\Sigma[T_1, T_2]$.

Let T_1 and T_2 be well-formed. Given a second-order unification problem $T_1 \doteq T_2$, then, following Pfenning [26], we construct a term of domain-free ML by the use of the following $\mathcal{UT}$ and $\mathcal{TI}$:

$$\mathcal{UT}(\Sigma[T_1, T_2]; T_1 \doteq T_2) = \lambda z_1.\lambda z_2.\lambda f.fz_1(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2))))),$$

where $\mathcal{TI}(\Sigma; z; T)$ is defined by induction on T:
(1) $\mathcal{TI}(\Sigma; z; t) = \lambda f.fz(fx(\lambda g.g))$ where $\Sigma(x) = t \in Con(T_1, T_2)$
(2) $\mathcal{TI}(\Sigma; z; X\tau_1 \cdots \tau_n) = \lambda f.fz(f(x[\tau_1] \cdots [\tau_n])(\lambda g.g))$
where $\Sigma(x) = \forall t_1 \cdots t_n.(Xt_1 \cdots t_n)$
(3) $\mathcal{TI}(\Sigma; z; T_1 \to T_2) = \lambda z_1.\lambda z_2.\lambda f.f(zz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2))))$

Remarks 1 *The reduction via $\mathcal{UT}$ and $\mathcal{TI}$ gives a β-normal term.*

The translation $\mathcal{TI}(\Sigma; z; T)$ says that type of z would be a substitution instance of T, see Lemma 5 below.

We next construct $ty(T)$ that is a type of $\mathcal{TI}(\Sigma; z; T)$. Although $ty(T)$ itself may not be a monotype, it becomes a monotype under some substitution if unifiable. Here of course we assume that we have countably many type variables to use a fresh type variable t for each application of the following definition:
(0) $ty(\tau) = (\tau \to (t \to t) \to t \to t) \to t \to t$ for $\tau \in Con(T)$;
(1) $ty(X\tau_1 \cdots \tau_n) = ((X\tau_1 \cdots \tau_n) \to (t \to t) \to t \to t) \to t \to t$ where $n \geq 0$;
(2) $ty(T_1 \to T_2) = T_1 \to T_2 \to (T_2 \to A \to A) \to A$
where $A \equiv (ty(T_1) \to ty(T_2) \to t) \to t$.

Lemma 5 $S(\Sigma[T]), x:\tau \vdash \mathcal{TI}(\Sigma[T]; x; T) : S(ty(T))$ *in domain-free* ML *if and only if* $S(T) =_\beta \tau$.

Proof. By induction on T. We show the following two cases:
(1) Case of $T \equiv X\tau_1 \cdots \tau_n$
Let $S(\Sigma(y))$ be $\forall t_1 \cdots t_n.((SX)t_1 \cdots t_n)$. We have
$S(\Sigma), x:\tau \vdash \lambda f.fx(f(y[\tau_1]\cdots[\tau_n])(\lambda g.g)) :$
$(((SX)\tau_1 \cdots \tau_n) \to (t \to t) \to t \to t) \to t \to t$
in domain-free ML. Here, types of x and $y[\tau_1]\cdots[\tau_n]$ must be equal. That is, $\tau =_\beta ((SX)\tau_1 \cdots \tau_n)$.
(2) Case of $T \equiv T_1 \to T_2$
From the definition, in domain-free ML we have
$S(\Sigma), x:\tau \vdash \lambda z_1.\lambda z_2.\lambda f.f(xz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2)))) :$
$S(T_1) \to S(T_2) \to (S(T_2) \to A \to A) \to A$
where $A \equiv (S(ty(T_1)) \to S(ty(T_2)) \to t) \to t$.
Then, we also have
$S(\Sigma), x:\tau, z_1:S(T_1), z_2:S(T_2) \vdash$
$\lambda f.f(xz_1)(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2)))) : (S(T_2) \to A \to A) \to A$.
Here, from the induction hypotheses, we have the following:
$S(\Sigma), z_1:\tau_3 \vdash \mathcal{TI}(\Sigma; z_1; T_1) : S(ty(T_1))$ iff $\tau_3 =_\beta S(T_1)$
$S(\Sigma), z_2:\tau_4 \vdash \mathcal{TI}(\Sigma; z_2; T_2) : S(ty(T_2))$ iff $\tau_4 =_\beta S(T_2)$
Now, type of (xz_1) and z_2 must be equal, i.e., $\tau =_\beta S(T_1) \to S(T_2)$. □

Lemma 6 (main lemma) $S(T_1) =_\beta S(T_2)$ *if and only if* $S(\Sigma) \vdash \mathcal{UT}(\Sigma; T_1 \doteq T_2) : S(ty(T_1 \to T_2))$ *in domain-free* ML.

Proof. $S(\Sigma) \vdash \mathcal{UT}(\Sigma; T_1 \doteq T_2) : S(ty(T_1 \to T_2))$
iff $S(\Sigma), z_1:S(T_1), z_2:S(T_2) \vdash \lambda f.fz_1(fz_2(\lambda g.g(\mathcal{TI}(\Sigma; z_1; T_1))(\mathcal{TI}(\Sigma; z_2; T_2))))$
$: (S(T_2) \to A \to A) \to A$ where $A \equiv (S(ty(T_1)) \to S(ty(T_2)) \to t) \to t$
iff (Lemma 5) $S(T_1) =_\beta S(T_2)$. □

Proposition 1 *The unification problem on the well-formed expressions is reduced to the problem of strong type inference for domain-free* ML. *In other words,* $S(T_1) =_\beta S(T_2) \iff \exists \Gamma.\exists \tau.\ \Gamma, \Gamma_0^{T_{1,2}} \vdash M^{T_{1,2}} : \tau$ *in domain-free* ML.

Proof. ($\Rightarrow$): From Lemma 6, $\Gamma_0^{T_{1,2}}$ and $M^{T_{1,2}}$ are determined by T_1 and T_2, such that for each $t \in Con(T_1, T_2)$, we have $\Gamma_0(x) = t$ for some x, and that $M = \mathcal{UT}(\Sigma; T_1 \doteq T_2)$. Then the unifier S gives Γ and τ, respectively, such that $S(\Sigma[T_1, T_2]) = \Gamma, \Gamma_0$ and $S(ty(T_1 \to T_2)) = \tau$.
($\Leftarrow$): Given $\Gamma_0^{T_{1,2}}$ and $M^{T_{1,2}}$, and assume that there exist τ and Γ such that $\Gamma = \{x_1 : \tau_1, \cdots, x_m : \forall t_1 \cdots t_n.\tau_m\}$. For each $X_i \in uVar(T_1, T_2)$, assume that $\Sigma(x_1) = X_1, \cdots, \Sigma(x_m) = \forall t_1 \cdots t_n.(X_m t_1 \cdots t_n)$. Then an answer to the trivial second-order matching problem such that $X_1 \doteq \tau_1, \cdots, (X_m t_1 \cdots t_n) \doteq \tau_m$ finds a matching S for $ty(T_1 \to T_2) \doteq \tau$, since if $S(\Sigma[T]), x:\tau_0 \vdash \mathcal{TI}(\Sigma[T]; x; T) : \tau$ for some τ, then $S(ty(T)) =_\beta \tau$. From Lemma 6, the unifier S is an answer to the unification problem $T_1 \doteq T_2$. □

Proposition 2 ($\mathrm{no}(\mathrm{nf},\mathrm{str})^{P}\mathbf{2}$) *The problem of strong type inference is undecidable for domain-free* ML*, even when the given term is a normal form.*

Proof. From Theorem 1, Proposition 1, and Remark 1. □

Remarks 2 ($\mathrm{no}(\mathrm{nf},\mathrm{str})^{R}\mathbf{2}$) *The proof of undecidability for strong type inference is also applicable to that for* DF&[ ] *terms in* ML*-fragment.*

Theorem 2 ($\mathrm{no}^{T}\mathbf{2}, \mathrm{no}(\mathrm{nf})^{T}\mathbf{2}$)**.** *Type checking, typability, and strong type inference are undecidable for domain-free* $\lambda 2$*.*

Proof. From Proposition 2 and Lemma 2. Moreover, even in the case where the given term is a normal form, typability and strong type inference for domain-free $\lambda 2$ are still undecidable. □

The problem of typability becomes undecidable for some predicative extension of the domain-free ML. We introduce a predicative fragment of domain-free $\lambda 2$, called domain-free ML_2. This extension allows us to abstract a term variable with a polymorphic type σ (polymorphic abstraction), but not to apply a polymorphic function to a polymorphic type (i.e., only to a monomorphic type τ). For this purpose, an extension of type schemes is introduced as follows:

$$\rho ::= \ \tau \mid \sigma \to \rho$$

This type ρ belongs to the so-called S(2)-class in [17], which is a special form of restrict types of rank 2 [20]. Following [17], a set of types with rank-k, denoted by $R(k)$ is defined as follows:
$R(0) =$ set of monotypes;
$R(k+1) = R(k) \mid R(k) \to R(k+1) \mid \forall t.R(k+1)$.

Domain-free (DF) ML_2-fragment:

$$\frac{\Gamma(x) = \forall t_1 \cdots t_n.\tau}{\Gamma \vdash x[\tau_1]\cdots[\tau_n] : \tau[t_1 := \tau_1, \cdots t_n := \tau_n]} \ (n \geq 0)$$

$$\frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2} \qquad \frac{\Gamma, x{:}\sigma \vdash M : \rho}{\Gamma \vdash \lambda x.M : \sigma \to \rho}$$

The problem of strong type inference in domain-free ML can be reduced to the typability problem in domain-free ML_2. That is, let $\{x_1, \cdots, x_n\}$ be a set of free variables in M; $\exists\sigma_1 \cdots \sigma_n.\exists\tau.\ \Gamma_0, x_1{:}\sigma_1, \cdots, x_n{:}\sigma_n \vdash M : \tau$ in DF ML-fragment if and only if $\exists\sigma_1 \cdots \sigma_n.\exists\tau.\ \Gamma_0 \vdash \lambda x_1 \cdots x_n.M : \sigma_1 \to \cdots \to \sigma_n \to \tau$ in DF ML_2-fragment. From Proposition2, we can obtain that the typability problem is undecidable for the rank 2 fragment of domain-free $\lambda 2$. This result means that the partial type reconstruction problem is still undecidable even for the rank 2 fragment of $\lambda 2$.

Corollary 1 ($\mathrm{no}(\mathrm{nf})^{C}\mathbf{1}$) *The typability problem for* DF ML_2 *is undecidable.*

Remarks 3 (no^R**3**) *From Lemma 2, the type checking problem for domain-free terms becomes undecidable at rank* 3.

Remarks 4 (no(nf)^R**4**) *From remark 2, the typability problem for* DF&[ ] *terms in* ML_2 *is undecidable.*

6 Static Properties of [ ]-Application

In this section, we study type inference, typability, and type checking with respect to [ ]-application terms. That is, the preterms have the following structure:
$P ::= x \mid \lambda x{:}\sigma.P \mid PP \mid \lambda t.P \mid P[\,]^a$
We first give a translation $\lceil\ \rceil$ from a term in Church-style to [ ]-application terms, which is motivated by the translation [5] from Church-style to domain-free style. Let $\mathtt{id} \equiv \lambda t.\lambda x{:}t.x$. Then $\mathtt{id} : \forall t.(t \to t)$.
$\lceil x \rceil = x; \quad \lceil \lambda x{:}\sigma.M \rceil = \lambda x{:}\sigma.\lceil M \rceil;$
$\lceil M_1 M_2 \rceil = \lceil M_1 \rceil \lceil M_2 \rceil; \quad \lceil \lambda t.M \rceil = \lambda t.\lceil M \rceil;$
$\lceil M[\sigma] \rceil = (\lambda v{:}\sigma \to \sigma.\lceil M \rceil[\,]^a)(\lambda x{:}\sigma.\mathtt{id}[\,]^a x)$ where v is a fresh variable.
For any term M in Church-style, $\lceil M \rceil$ is a preterm of [ ]-applications.

Proposition 3 *$\Gamma \vdash M : \sigma$ in Church-style $\lambda 2$ iff $\Gamma \vdash_p \lceil M \rceil : \sigma$.*

The problem of type inference for $\lambda 2$ in Church-style is reduced to the (strong) type inference problem for preterms of [ ]-applications. Hence from the undecidability of type inference for $\lambda 2$ in Church-style [29], the (strong) type inference problem for preterms of [ ]-applications is undecidable.

Theorem 3 (no^T**3**)**.** *The (strong) type inference problem for [ ]-application terms is undecidable.*

Remarks 5 (no^R**5**) *Following* [28]*, since the type inference problem is undecidable for* ML*-fragment in Church-style, the (strong) type inference problem for [ ]-application terms becomes undecidable in the* ML*-fragment.*

By the use of [ ]-applications, we give a translation which moves existentially quantified types in the context to the right-hand side of $\vdash_p$.

Proposition 4 *$\exists\sigma_1 \cdots \sigma_n.\exists\sigma.\ x_1{:}\sigma_1, \cdots, x_n{:}\sigma_n \vdash_p P : \sigma$ if and only if $\exists\sigma.\ y_1{:}\forall t.t, \cdots, y_n{:}\forall t.t \vdash_p \langle P \rangle : \sigma$, where $\langle x_i \rangle = y_i[\,]^{a_i}, \quad \langle x \rangle = x$ if $x \not\equiv x_i$ $(1 \le i \le n)$;*
$\langle \lambda x{:}\sigma.P \rangle = \lambda x{:}\sigma.\langle P \rangle; \quad \langle P_1 P_2 \rangle = \langle P_1 \rangle \langle P_2 \rangle;$
$\langle \lambda t.P \rangle = \lambda t.\langle P \rangle; \quad \langle P[\sigma] \rangle = \langle P \rangle[\sigma]; \quad \langle P[\,] \rangle = \langle P \rangle[\,].$

The variable x declared in Γ is translated to $y[\,]^a$, and the new variable y appears only in the form of $y[\,]^a$ in $\langle P \rangle$. The problem of type inference for [ ]-application terms is reduced to the typability problem for [ ]-application terms. From Propositions 3 and 4, the typability problem for [ ]-application terms is essentially equivalent to type inference in Church-style. From Theorem 3, the typability problem for [ ]-application terms becomes undecidable.

Theorem 4 (no^{T}**4**). *The typability problem for* [ ]*-application terms is undecidable.*

Remarks 6 (no^{R}**6**) *From Remark 5, the typability problem for* [ ]*-application terms becomes undecidable at rank* 2*. This rank* 2 *fragment is impredicative, i.e., cannot be the* ML*-fragment, since type schemes* (*rank* 1 *types*) *can be substituted.*

We next give a reduction from typability to type checking by the use of [ ]-applications.

Proposition 5 $\exists\sigma.\ \Gamma \vdash_p P : \sigma$ *if and only if* $\Gamma, x{:}\forall t_1.(t_1 \to t_2) \vdash_p x[\,]P : t_2$.

The problem of typability for [ ]-application terms is reduced to the type checking problem of [ ]-application terms. From Theorem 4, the type checking problem for [ ]-application terms becomes undecidable.

Theorem 5 (no^{T}**5**). *The problem of type checking for* [ ]*-application terms is undecidable.*

Remarks 7 (no^{R}**7**) *From Remark 6, the type checking problem for* [ ]*-application terms becomes undecidable at rank* 3*.*

Remarks 8 (no(nf)^{R}**8**) *From Remark 4 and Proposition 5, the type checking problem for* DF&[ ] *terms is undecidable at rank* 3*, even when the given preterm is a normal form.*

7 Related Work and Concluding Remarks

7.1 Related Work and Summary of Results

Relating to Proposition 2, the problem of strong type inference is also undecidable for domain-free ML with non-sorted variables [4], since the given proof with a slight modification still works for the definition of T, where type variable t is replaced with variable x, and

$\tau ::= x \mid \tau \to \tau$

by the use of a single syntactic category of variables x. This implies that strong type inference is undecidable for domain-free $\lambda 2$ with non-sorted variables.

Our result also implies a negative[2] answer to the question posed by Barthe and Sørensen [4] to know whether the problem of type checking is decidable for

[2] Following [5], type checking of domain-free $\lambda 2$ becomes decidable when the given term is in β-normal form. By the use of domain-erasing translation from Church-style to domain-free $\lambda 2$ inducing β-redexes, Barthe and Sørensen [5] proved that the type inference problem is undecidable for domain-free $\lambda 2$ with sorted variables. Although both works are based on [28], the distinction is that our reduction induces no redex, see Remark 1, and our method with a slight modification is also applicable to the strong type inference problem of domain-free $\lambda 2$ with non-sorted variables.

domain-free $\lambda 2$ and $\lambda\omega$. Moreover, the type checking problem for domain-free $\lambda\mu$-calculus introduced in [10] also becomes undecidable.

Besides already known ones, we summarize results obtained here, written in *italic* type marked with superscript. "no (nf)" means that the problem is undecidable even if the given term is a normal form, and "(str)" denotes the strong type inference problem. "yes (nf)" means that the problem becomes decidable only if the given term is a normal form.

For positive results, it is noted that Odersky and Läufer [22] have proposed a decidable extension of ML with *fully explicit* type scheme annotations so that the system can express polymorphic function arguments. See also Garrigue and Rémy [13] for a conservative extension of ML with *semi-explicit* first-class polymorphism.

Decidability of type inference, typability, and type checking for $\lambda 2$

$\lambda 2$	Church	Domain-Free	[ ]-App.	DF&[ ]	Curry
$? \vdash P :?$	no(nf)[28]	$no(nf, str)^{P}2$	$no^{T}3$	$no(nf, str)^{R}2$	no[32]
$\Gamma \vdash P :?$	yes[31]	$no(nf)^{T}2$	$no^{T}4$	$no(nf)^{R}4$	no[32]
$\Gamma \vdash P : \sigma?$	yes[31]	yes(nf)[5], $no^{T}2$	$no^{T}5$	$no(nf)^{R}8$	no[32]

Decidability of type inference

$? \vdash P :?$	Church	Domain-Free	[ ]-App.	DF&[ ]	Curry
$\lambda 2$	no(nf)	$no(nf, str)$	$no^{T}3$	$no(nf, str)$	no[32]
rank 2	no(nf)	$no(nf, str)$	no	$no(nf, str)$	yes(str)[17]
ML	no(nf)[28]	$no(nf, str)^{P}2$	$no^{R}5$	$no(nf, str)^{R}2$	yes[18]

yes(str)[17] and yes[17] below provided that the given context assigns closed type schemes to object variables.

Decidability of typability

$\Gamma \vdash P :?$	Church	Domain-Free	[ ]-App.	DF&[ ]	Curry
$\lambda 2$	yes[31]	$no(nf)^{T}2$	$no^{T}4$	$no(nf)$	no[32]
rank 2	yes	$no(nf)^{C}1$	$no^{R}6$	$no(nf)^{R}4$	yes[17]
ML	yes	yes	yes	yes	yes

The well-known $\mathcal{W}$ [21,9] is also applicable to the typability for ML in domain-free, [ ]-applications, and DF&[ ] cases. Remarked that typable terms at rank 1 coincide with typable terms in ML-fragment, and that rank 1 types are applied to type-directed compilation [23].

Decidability of type checking

$\Gamma \vdash P : \sigma$?	Church	Domain-Free	[ ]-App.	DF&[ ]	Curry
$\lambda 2$	yes[31]	yes(nf)[5], $no^T 2$	$no^T 5$	$no(nf)$	no
rank 3	yes	yes(nf), $no^R 3$	$no^R 7$	$no(nf)^R 8$	no[32]
rank 2	yes	yes(nf), ?	?	?	yes[19]
ML	yes	yes	yes	yes	yes

7.2 Final Observation and Concluding Remarks

It is worthwhile to mention the striking contrast between yes[18], yes(str)[17] and no(nf)[28], $no(nf, str)^P 2$, $no^R 5$, $no(nf, str)^R 2$. Pfenning [26] has proved that the partial type reconstruction problem is undecidable for a predicative fragment of $\lambda 2$. However, the undecidability has been explicitly discussed neither from the viewpoint of rank nor type information of domains and holes [27]. Our result shows that existence of type information makes type inference problems undecidable for ML-fragment, in contrast with the case of Curry-style [18]. From this, the partial type reconstruction problem becomes undecidable for the rank 2 fragment of $\lambda 2$, and moreover making polymorphic domains free and the use of type-holes [ ] are independently responsible for the undecidability. The distinction between yes and no for ML-fragment can be observed in the following.

Given a term M in Church-style ML. Then consider a problem to find a context and a type such that $? \vdash M :?$ holds in Church-style ML. First consider the typability problem of $\|\,|M|\,\|$ in Curry-style ML. Let $\{x_1, \cdots, x_m\}$ be a set of free variables in M. Let Γ be $x_1 : \forall t.t, \cdots, x_m : \forall t.t$. If $\|\,|M|\,\|$ is typable, then we have the principal type τ of $\|\,|M|\,\|$, such that $\mathcal{W}(\Gamma, \|\,|M|\,\|) = (S, \tau)$ and $\Gamma \vdash \|\,|M|\,\| : \tau$ in Curry-style ML. If $\|\,|M|\,\|$ is not typable, then M is not typable in Church-style, see (1) and (2) in Section 3.

Assume that $\|\,|M|\,\|$ is typable. Let x be a free variable appeared k times in M ($k \geq 1$). From the derivation of $\Gamma \vdash \|\,|M|\,\| : \tau$ in Curry-style ML, we obtain principal monotypes for each occurrence of x in $\|\,|M|\,\|$, say $x : \tau_1, \cdots, x : \tau_k$. Without loss of generality, assume that x appear as one of the following forms in M:
(1) x is used polymorphically in M, such that

$$\begin{cases} x[\tau_{11}][\tau_{12}] \cdots [\tau_{1n}] \\ \cdots\cdots \\ x[\tau_{k1}][\tau_{k2}] \cdots [\tau_{kn}] \end{cases}$$

(2) k occurrences of x have no type applications in M.
In order to solve the original type inference problem ($? \vdash M :?$ in Church-style ML), we have to check that type of the given term M can be the same as the inferred type τ of the erasure, see also (-1) and (-2) in Section 3, and then at least

we have to find an answer to the following second-order unification problem:
Case of (1)

$$\begin{cases} F\tau_{11}\tau_{12}\cdots\tau_{1n} \doteq \tau_1 \\ \qquad\cdots\cdots \\ F\tau_{k1}\tau_{k2}\cdots\tau_{kn} \doteq \tau_k \end{cases}$$

where F is an unknown second-order variable with n-arity, in which every argument of F contains no variables for the unification problem. Type variables in $\tau_1, \cdots, \tau_k$ are used as unknown first-order variables for the unification. This unification problem is a variant of the problem in Section 5, i.e., the simple instances of the second-order unification problem [28,29] with first-order variables. The case of (2) gives the first-order unification problem:

$\tau_1 \doteq \tau_2 \doteq \cdots \doteq \tau_k$

Following the observation above, $\mathcal{W}$ *plus* an answer to the second-order unification problem could solve the problems of type inference in the cases of Church, domain-free, [ ]-applications, and DF&[ ], i.e., additional type information gives more constraints to be solved.

Another observation is about the definition of preterms. Alternatively, one can define preterms with full annotations, such that
$Q ::= x \mid \lambda x{:}[\sigma]^a.Q \mid QQ \mid \lambda t.Q \mid Q[\sigma]^a \mid \lambda x{:}[\,]^a.Q \mid Q[\,]^a$
Then $\Gamma \vdash_p Q : \sigma; \Delta$ can be similarly defined, and the same results in 7.1 also hold for the preterms following a similar pattern.

Acknowledgements
We are grateful to J. Roger Hindley and Horai-Takahashi Masako for helpful discussions. We would like to thank Frank Pfenning for valuable comments on this work, and Gilles Barthe and Morten Heine B. Sørensen for pointing out their paper in the revised version [5]. The authors are also grateful to anonymous referees for their constructive comments and suggestions.

References

1. Barendregt, H. P.: Lambda Calculi with Types, Handbook of Logic in Computer Science Vol.II, Oxford University Press, pp. 1–189, 1992.
2. Boehm, Hans-J.: Partial Polymorphic Type Inference is Undecidable, *Proc. 26th Annual Symposium of Fundations of Computer Science*, pp. 339–345, 1985.
3. Barthe, G., Hatcliff, J., and Sørensen, M. H.: CSP Translations and Applications: The Cube and Beyond, *Proc. ACM SIGPLAN on Continuations*, pp. 1–31, 1996.
4. Barthe, G. and Sørensen, M. H.: Domain-Free Pure Type Systems, Lecture Notes in Computer Science 1234, pp. 9–20, 1997.
5. Barthe, G. and Sørensen, M. H.: Domain-Free Pure Type Systems.
6. Church, A.: A Formulation of the Simple Theory of Types, *The Journal of Symbolic Logic*, Vol. 5, pp. 56–68, 1940.
7. Curry, H. B.: Functionality in combinatory logic, *Proc. National Academy of Sciences of the USA* 20, pp. 584–590, 1934.

8. Curry, H. B., Feys, R. and Craig, W.: Combinatory Logic, Volume 1 (Third printing), North-Holland, 1974.
9. Damas, L. and Milner, R.: Principal type-schemes for functional programs, *Proc. ACM Symposium on Principles of Programming Languages*, pp. 207–212, 1982.
10. Fujita, K.: Explicitly Typed $\lambda\mu$-Calculus for Polymorphism and Call-by-Value, Lecture Notes in Computer Science 1581, pp. 162–176, 1999.
11. Fujita, K.: Type Inference for Domain-Free $\lambda 2$, Technical Report in Computer Science and Systems Engineering CSSE–5, Kyushu Institute of Technology, 1999.
12. Fujita, K: Undecidability of Partial Type Reconstruction, Computer Software, Vol. 17, No. 2, pp. 40–44, 2000 (*in Japanese*).
13. Garrigue, J. and Rémy, D.: Semi-Explicit First-Class Polymorphism for ML, *Information and Computation*, 1999.
14. Harper, R. and Lillibridge, M.: ML with callcc is unsound, The Types Form, 8 July 1991.
15. Harper, R. and Mitchell, J. C.: On The Type Structure of Standard ML, *ACM Trans. on Programming Languages and Systems*, Vol. 15, No. 2, pp. 210–252, 1993.
16. Hindley, J. R.: Basic Simple Type Theory, Cambridge University Press, 1997.
17. Kfoury, A. J. and Tiuryn, J. : Type Reconstruction in Finite Rank Fragments of the Second-Order λ-Calculus, *Information and Computation* 98, pp. 228–257, 1992.
18. Kfoury, A. J., Tiuryn, J. and Urzyczyn, P.: An Analysis of ML Typability, *Journal of the Association for Computing Machinery*, Vol. 41, No. 2, pp. 368–398, 1994.
19. Kfoury, A. J. and Wells, J. B.: A direct algorithm for type inference in the rank-2 fragment of the second-order λ-calculus, *Proc. ACM LISP and Functional Programming*, pp. 196–207, 1994.
20. Leivant, D.: Finitely Stratified Polymorphism, *Information and Computation* 93, pp. 93–113, 1991.
21. Milner, R.: A Theory of Type Polymorphism in Programming, *Journal of Computer and System Sciences* 17, pp. 348–375, 1978.
22. Odersky, M. and Läufer, K.: Putting Type Annotations to Work, *Proc. ACM Symposium on Principles of Programming Languages*, pp. 54–76, 1996.
23. Ohori, A. and Yoshida, N.: Type Inference with Rank 1 Polymorphism for Type-Directed Compilation for ML, *Proc. ACM Conference on Functional Programming*, pp. 160–171, 1999.
24. Parigot, M.: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science 624, pp. 190–201, 1992.
25. Pfenning, F.: Partial polymorphic type inference and higher-order unification, *Proc. ACM Conference on Lisp and Functional Programming*, pp. 153–163, 1998.
26. Pfenning, F.: On the undecidability of partial polymorphic type reconstruction, *Fundamenta Informaticae* 19, pp. 185–199, 1993.
27. Pfenning, F.: private communication, 9 June 1999.
28. Schubert, A.: Second-order unification and type inference for Church-style, Tech. Report TR 97-02 (239), Institute of Informatics, Warsaw University, January 1997.
29. Schubert, A.: Second-order unification and type inference for Church-style, *Proc. ACM Symposium on Principles of Programming Languages*, pp. 279–288, 1998.
30. Tiuryn, J.: Type Inference Problems: A Survey, Lecture Notes in Computer Science 452, pp. 105–120, 1990.
31. Van Benthem Jutting, L. S.: Typing in Pure Type Systems, *Information and Computation* 105, pp. 30–41, 1993.
32. Wells, J. B.: Typability and Type Checking in the Second-Order λ-Calculus Are Equivalent and Undecidable, *Proc. IEEE Symposium on Logic in Computer Science*, pp. 176–185, 1994.

Alternating Automata and Logics over Infinite Words
(Extended Abstract)

Christof Löding and Wolfgang Thomas

Lehrstuhl für Informatik VII, RWTH Aachen, D-52056 Aachen
{loeding,thomas}@informatik.rwth-aachen.de

Abstract. We give a uniform treatment of the logical properties of alternating weak automata on infinite strings, extending and refining work of Muller, Saoudi, and Schupp (1984) and Kupferman and Vardi (1997). Two ideas are essential in the present set-up: There is no acyclicity requirement on the transition structure of weak alternating automata, and acceptance is defined only in terms of reachability of states; moreover, the run trees of the standard framework are replaced by run dags of bounded width. As applications, one obtains a new normal form for monadic second order logic, a simple complementation proof for weak alternating automata, and elegant connections to temporal logic.

1 Introduction

Finite automata on infinite strings provide a useful framework for the logical analysis of sequence properties. The connection to logic is based on (at least) the following four aspects:

- Nondeterministic Büchi automata are expressively equivalent to monadic second-order logic (MSO-logic) over infinite strings ([Büc62]). This equivalence involves a normal form of MSO-formulas in EMSO-logic (existential monadic second-order logic).
- Connected with this fact is the closure of Büchi automata under complement.
- A hierarchy of acceptance conditions for deterministic ω-automata induces a natural classification of sequence properties (cf. [MP92]), including, for example, safety properties and recurrence properties.
- Propositional temporal logic PLTL, a standard framework for the specification of infinite computations, is characterized by counter-free deterministic Muller automata, defined by a natural restriction on the loop structure of transition graphs.

In the present paper, we study these logical aspects of ω-languages in the framework of alternating weak automata, a model introduced in the pioneering work of Muller, Saoudi, and Schupp [MSS86]. We introduce a variant of alternating weak automata which differs from the model of [MSS86] in the following way:

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 521–535, 2000.
© Springer-Verlag Berlin Heidelberg 2000

There is no acyclicity requirement on the transition structure of weak alternating automata, and acceptance is defined only in terms of reachability of states; moreover, the run trees of the standard framework are replaced by run dags of bounded width. (For the equivalence to the model of [MSS86] see the next section.) Starting from this, it turns out that all four aspects mentioned above have counterparts in the framework of alternating automata:

1. The equivalence between alternating weak automata and monadic second-order logic over infinite strings provides a new normal form of MSO-formulas, giving an alternative to the classical EMSO-normal form (for the specification of accepting runs).
2. The complementation of alternating weak automata is presented in a game theoretic setting, based on a determinacy result on infinite games with winning conditions in terms of reachability of states. (For a separate exposition of this result see [Tho99].)
3. The basic classification of temporal properties (called Landweber hierarchy in the automaton theoretic setting) is captured in the framework of alternating automata in two different ways: by restricting the alternating computation mode (to universal, respectively existential branching) and by restricting the acceptance component in alternating automata.
4. A structural property (on the loop structure) of weak alternating automata is presented, which characterizes the properties definable in the temporal logic PLTL.

As mentioned above, we define acceptance by alternating automata using run dags instead of run trees. In [KV97], alternating automata are defined as in [MSS86] (however with the co-Büchi acceptance condition), and then a reduction to acceptance via run dags is carried out. In both cases, the approach via run dags does not weaken the expressive power due to the fact that in the associated infinite games (see Section 3) memoryless winning strategies are sufficient. The determinacy proof presented in this paper (of which a preliminary exposition was given by the second author in [Tho99]) is related to a construction of Klarlund [Kla91]. The structural characterization of PLTL-definable properties was obtained independently by Rohde [Roh97], however with a more involved proof.

The paper is structured as follows: In Section 2 we introduce alternating automata and their acceptance conditions. In Section 3, the dualization of alternating automata and its connection to determinacy of infinite games and to complementation is developed (see item 2 above). Section 4 shows that alternating weak automata are able to recognize precisely the regular ω-languages, via a transformation of parity automata into alternating weak automata. Finally, Sections 5, 6, and 7 present the results mentioned above under item 1 (connection to MSO-logic), item 3 (classification of sequence properties) and item 4 (characterization of PLTL-definable properties).

2 Alternating Automata

Alternating automata combine the possibility of existential and universal branching. The transition function of an alternating automaton is defined with positive boolean formulas over the state set.

Let X be a finite set. The set of positive boolean formulas over X, denoted by $\mathcal{B}^+(X)$, contains $\top$ (true), $\bot$ (false), all elements from X, and all boolean combinations over X built with $\wedge$ and $\vee$. A subset S of X is a *model* of $\theta \in \mathcal{B}^+(X)$ iff the truth assignment that assigns true to the elements of S and false to the elements of $X \setminus S$ satisfies θ. We say S is a *minimal model* of θ iff S is a model of θ and no proper subset of S is a model of θ. For $\theta \in \mathcal{B}^+(X)$ the set of minimal models of θ is denoted by $\mathcal{M}_\theta$.

An alternating automaton $\mathcal{A}$ is of the form $\mathcal{A} = (Q, \Sigma, q_0, \delta, AC)$, where Q is a finite state set, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$ is the transition function and AC is the acceptance component. There are several different types of acceptance conditions referring to different types of acceptance components.

Since in an alternating automaton there is universal branching, a run of an alternating automaton is not an infinite sequence of states, but an infinite acyclic graph. This graph has a "root vertex" labelled with the initial state q_0 and in distance l from this vertex one finds the states which are assumed by the automaton after l input letters. Formally a run is defined as follows. Let $\alpha \in \Sigma^\omega$ and let $G = (V, E)$ be a directed acyclic graph with the following properties.

- $V \subseteq Q \times \mathbb{N}$ with $(q_0, 0) \in V$.
- $E \subseteq \bigcup_{l>0} (Q \times \{l\}) \times (Q \times \{l+1\})$.
- For every $(q, l) \in V \setminus \{(q_0, 0)\}$ exists a $q' \in Q$, such that $((q', l-1), (q, l)) \in E$.

G is called a *run* of $\mathcal{A} = (Q, \Sigma, q_0, \delta, A)$ on α, if for every $(q, l) \in V$ the set $\{q' \in Q \mid ((q, l), (q', l+1)) \in E\}$ is a minimal model of $\delta(q, \alpha(l))$. An example is given in Figure 1.

Note that there is no run $G = (V, E)$ on α, such that $(q, l) \in V$ for a q with $\delta(q, \alpha(l)) = \bot$, since $\bot$ has no models.

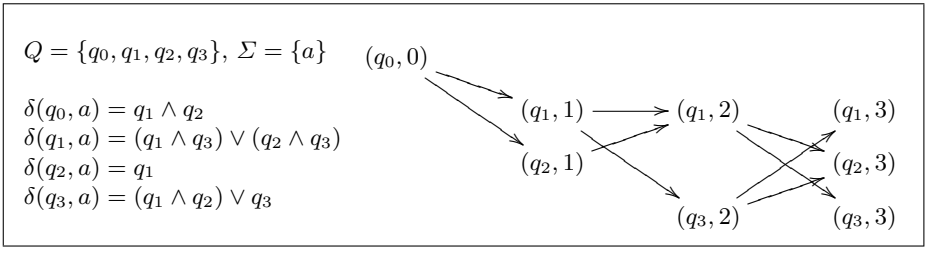


Fig. 1. First segment of a run of an alternating automaton.

With this definition of the transition function of alternating automata we get deterministic, nondeterministic, and universal automata as special cases of alternating automata. In deterministic automata the formulas used in the transition function consist of exactly one state. In nondeterministic automata the formulas are disjunctions of states or $\bot$, and dual to that, in universal automata the formulas are conjunctions of states or $\top$.

For later use in Section 7, we note that it is possible to use alternating automata with an initial positive boolean formula θ_0 instead of a single initial state. Such an automaton can be converted into an equivalent automaton with a single initial state just by adding one extra state.

An automaton accepts a word iff there exists a run of the automaton on this word such that every infinite path through that run satisfies the acceptance condition. The language accepted by the automaton consists of all the words that are accepted by the automaton. Here we identify an infinite path π with the sequence of states induced by this path. The *infinity set* $In(\pi)$ consists of all states that appear infinitely often in π. The *occurrence set* $Oc(\pi)$ consists of all states that appear at least once in π. The following different types of acceptance conditions are considered in this paper.

In *Büchi* and *co-Büchi automata* the acceptance condition refers to a subset F of the state set and in *parity automata* the acceptance condition refers to a mapping (called coloring) $c : Q \to \{0, \ldots, k\}$. The numbers $0, \ldots, k$ are called the colors. For an infinite path π the corresponding infinite sequence of colors is then denoted as $c(\pi)$. An infinite path π satisfies

- the Büchi condition w.r.t. F iff $F \cap In(\pi) \neq \emptyset$,
- the co-Büchi condition w.r.t. F iff $F \cap In(\pi) = \emptyset$,
- the parity condition w.r.t. c iff $\min(In(c(\pi)))$ is even.

For all of these acceptance types we can also consider the "weak versions". We call an acceptance condition weak if it is evaluated in the occurrence set instead of the infinity set of a path. So an infinite path π satisfies

- the weak Büchi condition w.r.t. F iff $F \cap Oc(\pi) \neq \emptyset$,
- the weak co-Büchi condition w.r.t. F iff $F \cap Oc(\pi) = \emptyset$,
- the weak parity condition w.r.t. c iff $\min(Oc(c(\pi)))$ is even.

In the present paper, we focus on weak automata with the weak parity acceptance condition. This differs from the model of weak automata as introduced by Muller and Schupp in [MSS86] where the Büchi acceptance condition is used. Moreover the transition structure of a weak automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$ must fulfill the following requirement: There is a partition $Q_1, \ldots, Q_m$ of Q such that for every $q \in Q_i$ and $q' \in Q_j$ with a transition leading from q to q' one has $j \leq i$, and $Q_i \subseteq F$ or $Q_i \cap F = \emptyset$ for every $i \in \{1, \ldots, m\}$.

Let us verify that the two models are equivalent in expressive power. Given a weak automaton $\mathcal{A}$ as above (in the sense of [MSS86]), acceptance means that for every path π through a run of $\mathcal{A}$ there is an $i \in \{1, \ldots, m\}$ such that $In(\pi) \subseteq Q_i \subseteq F$. This can also be expressed as a weak parity condition because

a path through the run satisfies the acceptance condition if it enters one of the accepting Q_i's and never leaves it again. For all $i \in \{1, \ldots, m\}$ and for all $q \in Q_i$ we define the coloring c as

$$c(q) = \begin{cases} 2i & \text{if } Q_i \subseteq F, \\ 2i+1 & \text{if } Q_i \cap F = \emptyset. \end{cases}$$

It is easy to see that the weak parity automaton $\mathcal{A}' = (Q, \Sigma, q_0, \delta, c)$ is equivalent to $\mathcal{A}$.

Conversely, given a weak parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ with $c : Q \to C$, let C_e be the set of even numbers in C and let $Q' = Q \times C$, $q'_0 = (q_0, c(q_0))$, $F' = Q \times C_\mathrm{e}$. To define the transition function we need an auxiliary mapping $\phi : \mathcal{B}^+(Q) \times C \to \mathcal{B}^+(Q')$. The formula $\phi(\theta, i)$ is obtained by replacing every $q \in Q$ in θ with $(q, \min\{i, c(q)\})$. Then we define the transition function by $\delta'((q,i),a) = \phi(\delta(q,a), i)$. In the second component of the new states the automaton remembers the minimal color that was seen so far. Along a path through a run this color may not increase. Therefore, if we define the sets Q_i according to the color in the second component, we get an automaton in the form of [MSS86] equivalent to $\mathcal{A}$.

3 Complementation

In [MS87] Muller and Schupp show that complementation of alternating automata can be done by dualization. In this section we give a self-contained proof of this complementation theorem for the case of alternating weak parity automata. This is done in a game theoretic framework, and making use of the simple winning conditions which are derived from weak alternating automata. So we do *not* rely on difficult determinacy results, e.g. for Borel games, as done in [MS87]. Before we turn to games we define the dual of an alternating weak parity automaton.

For a finite set X and $\theta \in \mathcal{B}^+(X)$ the formula $\widetilde{\theta}$, the dual of θ, is obtained by exchanging $\vee$ and $\wedge$, and $\bot$ and $\top$. We can state the following relation between the minimal models of θ and the models of $\widetilde{\theta}$.

Remark 1. Let $\theta \in \mathcal{B}^+(X)$. A set $S \subseteq X$ is a model of $\widetilde{\theta}$ iff $S \cap R \neq \emptyset$ for all minimal models R of θ.

Proof. The formula $\widetilde{\theta}$ is equivalent to $\bigwedge_{R \in \mathcal{M}_\theta} \bigvee_{x \in R} x$, which is the conjunctive normal form of $\widetilde{\theta}$. S is a model of $\widetilde{\theta}$ iff it contains at least one element from each of the disjunctive terms.

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton. The *dual automaton* $\widetilde{\mathcal{A}}$ of $\mathcal{A}$ is defined as $\widetilde{\mathcal{A}} = (Q, \Sigma, q_0, \widetilde{\delta}, \widetilde{c})$, where $\widetilde{\delta}$ is defined by $\widetilde{\delta}(q,a) = \widetilde{\delta(q,a)}$ for all $q \in Q$ and $a \in \Sigma$, and $\widetilde{c}$ is defined by $\widetilde{c}(q) = c(q) + 1$.

Since in $\widetilde{\mathcal{A}}$ a state has an even color iff it has an odd color in $\mathcal{A}$, we get the following remark.

Remark 2. A path π satisfies the acceptance condition of $\mathcal{A}$ iff it does not satisfy the acceptance condition of $\widetilde{\mathcal{A}}$.

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton and let $\alpha \in \Sigma^\omega$. With $\mathcal{A}$ and α we associate a graph $G_{\mathcal{A},\alpha} = (V_A, V_P, E, w)$ which serves as a game arena for the two players Automaton (A) and Pathfinder (P). The graph is defined as follows.

- $V_A = Q \times \mathbb{N}$ and $V_P = Q \times (2^Q \setminus \{\emptyset\}) \times \mathbb{N}$; let V always denote $V_A \cup V_P$.
- The edge relation E is defined by

$$\begin{array}{ll} ((q,l),(q,S,l+1)) \in E & \text{iff } \ S \in \mathcal{M}_{\delta(q,\alpha(l))}, \\ ((p,S,l),(q,l)) \in E & \text{iff } \ q \in S, \end{array}$$

  for $p, q \in Q$, $S \subseteq Q$ and $l \in \mathbb{N}$.
- The coloring $w : V \to c(Q)$ is given by $w((q,l)) = c(q)$ for $(q,l) \in V_A$ and $w((q,S,l)) = c(q)$ for $(q,S,l) \in V_P$.

A play of $G_{\mathcal{A},\alpha}$ is an infinite sequence $\gamma \in (V_A V_P)^\omega$ such that $\gamma(0) = (q_0, 0)$ and $(\gamma(i), \gamma(i+1)) \in E$ for all $i \in \mathbb{N}$. Automaton wins the play γ iff $\min(Oc(w(\gamma)))$ is even.

A positional strategy for A is a mapping $f_A : V_A \to V_P$ such that for all $v \in V_A$ we have $(v, f_A(v)) \in E$. The play γ is played according to f_A iff for every $i \in \mathbb{N}$ with $\gamma(i) \in V_A$ one has $\gamma(i+1) = f_A(\gamma(i))$. The strategy f_A is called a positional winning strategy for A iff A wins every play γ played according to f_A.

Strategies for P are defined analogously.

The connections between alternating weak parity automata and the corresponding games are stated in the following three lemmas.

Lemma 1. *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton and let $\alpha \in \Sigma^\omega$. Automaton has a positional winning strategy in $G_{\mathcal{A},\alpha}$ iff $\alpha \in L(\mathcal{A})$.*

The very simple proof is omitted.

Lemma 2. *(Determinacy of weak parity games) Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton and let $\alpha \in \Sigma^\omega$. In $G_{\mathcal{A},\alpha}$ either Automaton or Pathfinder has a positional winning strategy.*

Proof. We first define the notion of an attractor. Let $T \subseteq V_A \cup V_P$. The A-attractor of T, denoted $Attr_A(T)$, is the set of vertices from which Automaton can force the play to eventually visit T.

$$\begin{array}{rl} Attr_A(T) = & \bigcup_{i \in \mathbb{N}} Attr_A^i(T), \text{ where } Attr_A^0(T) = T \text{ and} \\ v \in Attr_A^{i+1}(T) \Leftrightarrow & v \in Attr_A^i(T) \text{ or} \\ & v \in V_A \text{ and } \exists (v,u) \in E \ : \ w \in Attr_A^i(T) \text{ or} \\ & v \in V_P \text{ and } \forall (v,u) \in E \ : \ w \in Attr_A^i(T). \end{array}$$

The P-attractor of T, denoted $Attr_P(T)$, is defined in the analogous way.

By induction on $m = |c(Q)|$, i.e., the number of colors in the weak automaton, we show that either of the players has a positional winning strategy. For $m = 1$ every play in $G_{\mathcal{A},\alpha}$ is won by Automaton or every play is won by Pathfinder. Therefore Automaton or Pathfinder has a positional winning strategy.

Let $m \geq 2$, $k = \min(w(V))$, and $V_k = \{v \in V \mid w(v) = k\}$. We assume that k is even. The proof for the other case is analogous.

If $(q_0, 0)$ belongs to $Attr_A(V_k)$, then obviously Automaton has a positional winning strategy in $G_{\mathcal{A},\alpha}$. If $(q_0, 0)$ does not belong to $Attr_A(V_k)$, then we define the game $G'_{\mathcal{A},\alpha}$ by removing the vertices of $Attr_A(V_k)$ from $G_{\mathcal{A},\alpha}$. By induction we know that in $G'_{\mathcal{A},\alpha}$ either Automaton or Pathfinder has a positional winning strategy. If Pathfinder has a winning strategy in $G'_{\mathcal{A},\alpha}$, then playing according to this strategy also forces the game to stay outside of $Attr_A(V_k)$ in the game $G_{\mathcal{A},\alpha}$. Otherwise there would be a vertex belonging to $Attr_A(V_k)$ in $G'_{\mathcal{A},\alpha}$. Therefore Pathfinder also has a positional winning strategy in $G_{\mathcal{A},\alpha}$.

Now suppose Automaton has a positional winning strategy in $G'_{\mathcal{A},\alpha}$. If Automaton plays according to this strategy in $G_{\mathcal{A},\alpha}$, then the only possibility for Pathfinder to give the play another progression as in $G'_{\mathcal{A},\alpha}$, is to move into $Attr_A(V_k)$ if possible. But then Automaton wins by forcing the game to move into V_k. Therefore Automaton has a positional winning strategy in $G_{\mathcal{A},\alpha}$.

Lemma 3. *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton and let $\alpha \in \Sigma^\omega$. Automaton has a positional winning strategy in $G_{\mathcal{A},\alpha}$ iff Pathfinder has a positional winning strategy in $G_{\widetilde{\mathcal{A}},\alpha}$.*

Proof. Let f_A be a positional winning strategy for Automaton in $G_{\mathcal{A},\alpha}$ and let (q, S, l) be a vertex of Pathfinder in $G_{\widetilde{\mathcal{A}},\alpha}$. If there exists a play such that (q, S, l) appears in this play, then $S \in \mathcal{M}_{\widetilde{\delta(q,\alpha(l-1))}}$. For vertices that may not appear in a play we do not have to define the strategy. From Remark 1 it follows that $S \cap f_A(q, l-1) \neq \emptyset$. Let $p \in S \cap f_A(q, l-1)$. We define $\widetilde{f}_P(q, S, l) = p$. For a play γ of $G_{\widetilde{\mathcal{A}},\alpha}$ played according to $\widetilde{f}_P$ there exists a play γ' of $G_{\mathcal{A},\alpha}$ played according to f_A such that $\widetilde{w}(\gamma) = w(\gamma')$. Since Automaton wins γ' in $G_{\mathcal{A},\alpha}$, Pathfinder wins γ in $G_{\widetilde{\mathcal{A}},\alpha}$.

Let $\widetilde{f}_P$ be a positional winning strategy for Pathfinder in $G_{\widetilde{\mathcal{A}},\alpha}$ and let (q, l) be a vertex of Automaton in $G_{\mathcal{A},\alpha}$. The set $S = \{\widetilde{f}_P(q, R, l) \mid R \in \mathcal{M}_{\widetilde{\delta(q,\alpha(l))}}\}$ is a model of $\delta(q, \alpha(l))$ by Remark 1. Let $S' \subset S$ be a minimal model of $\delta(q, \alpha(l))$. We define $f_A(q, l) = (q, S', l+1)$. Again, for a play γ in $G_{\mathcal{A},\alpha}$ played according to f_A there exists a play γ' played according to $\widetilde{f}_P$ in $G_{\widetilde{\mathcal{A}},\alpha}$ such that $w(\gamma) = \widetilde{w}(\gamma')$. Since Pathfinder wins γ' in $G_{\widetilde{\mathcal{A}},\alpha}$, Automaton wins γ in $G_{\mathcal{A},\alpha}$.

Theorem 1. *Let $\mathcal{A}$ be a alternating weak parity automaton over the alphabet Σ. Then $L(\widetilde{\mathcal{A}}) = \Sigma^\omega \setminus L(\mathcal{A})$.*

Proof. Let $\alpha \in \Sigma^\omega$. The automaton $\mathcal{A}$ accepts α iff Automaton has a positional winning strategy in $G_{\mathcal{A},\alpha}$, by Lemma 1. By Lemma 3 this is equivalent to Pathfinder having a winning strategy in $G_{\widetilde{\mathcal{A}},\alpha}$. With Lemma 2 we know that this is the same as Automaton having no positional winning strategy in $G_{\widetilde{\mathcal{A}},\alpha}$ and again using Lemma 1 this is equivalent to $\alpha \notin L(\widetilde{\mathcal{A}})$.

4 Expressive Completeness

In this section we show that every regular ω-language can be recognized by an alternating weak parity automaton. We give a transformation of deterministic parity automata into alternating weak parity automata; it seems to be the simplest way to establish expressive completeness of alternating weak parity automata. (It is well known that deterministic parity automata recognize precisely the regular ω-languages, see [Tho97]). In contrast to deterministic parity automata, where it is not possible to bound the number of colors, for alternating weak parity automata it suffices to consider automata with only three colors.

Theorem 2. *For every deterministic parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ with $|Q| = n$ and $c : Q \to \{1, \ldots, m\}$ one can construct an equivalent alternating weak parity automaton $\mathcal{A}' = (Q', \Sigma, q_0', \delta', c')$ with $|Q'| = (m+1)n$ and $c' : Q' \to \{1, 2, 3\}$.*

Proof. We can assume that $m = 2k$ for some k. Define $\mathcal{A}'$ as follows.

- Let $Q' = Q \cup (Q \times \{1, \ldots, k\} \times \{0, 1\})$ and $q_0' = q_0$.
- For $q \in Q$, $i \in \{1, \ldots, k\}$, $a \in \Sigma$, and $p = \delta(q, a)$ define

$$\delta'(q, a) = p \vee \bigvee_{j=1}^{k} (p, j, 0),$$

$$\delta'((q, i, 0), a) = \begin{cases} \bot \text{ if } c(q) < 2i, \\ (p, i, 0) \wedge (p, i, 1) \text{ otherwise,} \end{cases}$$

$$\delta'((q, i, 1), a) = \begin{cases} \top \text{ if } c(q) = 2i, \\ (p, i, 1) \text{ otherwise.} \end{cases}$$

- For $q \in Q$ and $i \in \{1, \ldots, k\}$ let $c'(q) = 3$, $c'((q, i, 0)) = 2$, $c'((q, i, 1)) = 1$.

The idea is to guess the accepting color and the point from where on no smaller color occurs, and then to verify that the guessed color occurs infinitely often and no smaller color occurs anymore. The correctness proof is omitted.

Note that along each path through a run of $\mathcal{A}'$ the colors are decreasing, which corresponds to the usual definition of weak automata. In fact our model without this restriction is equivalent to the usual model.

It is also possible to start from nondeterministic Büchi automata instead of deterministic parity automata, using a similar construction as in [KV97].

5 Alternating Automata and MSO

In this section we give a characterization of alternating weak parity automata in MSO logic. We obtain a normal form of MSO formulas different from the EMSO formulas obtained by Büchi [Büc62].

The MSO formulas we consider are S1S (second-order theory of one successor) formulas, built up in the usual way from $x, y, z, x_1, x_2, \ldots$ as (first-order) variables for natural numbers, $X, Y, Z, X_1, X_2, \ldots$ as (second-order) variables for sets of natural numbers, the symbols $0, +1, =, <, \in$ with their usual meanings, the connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and the quantifiers $\exists, \forall$.

We can interpret the sets of natural numbers as predicates, and abbreviate $x \in X$ with Xx. We also use natural abbreviations as $\leq, >, \geq, +2, +3, \ldots$ and $\exists^\omega$ ("there exist infinitely many"), $\forall^\omega$ ("for almost all").

The formulas are interpreted in the structure $(\mathbb{N}, <, +1, 0)$. As models we take tuples of subsets of $\mathbb{N}$. Such a tuple $\underline{\alpha} = (Q_1, \ldots Q_n)$, with $Q_1, \ldots, Q_n \subseteq \mathbb{N}$, is a model of a formula $\phi(X_1, \ldots X_n)$, denoted by $\underline{\alpha} \models \phi(X_1, \ldots X_n)$, if and only if ϕ evaluates to true when we substitute each X_i by Q_i.

To characterize automata in S1S, we use a correspondence between infinite words α over an alphabet and the models $\underline{\alpha}$ of S1S. Then we will construct a formula $\phi_{\mathcal{A}}$ such that the automaton $\mathcal{A}$ accepts a word α if and only if the corresponding tuple $\underline{\alpha}$ is a model of $\phi_{\mathcal{A}}$. In addition we construct a second formula $\widetilde{\phi}_{\mathcal{A}}$ equivalent to $\phi_{\mathcal{A}}$, such that $\neg\widetilde{\phi}_{\mathcal{A}} = \phi_{\widetilde{\mathcal{A}}}$. The connection between these two formulas corresponds to the connection between an automaton and its dual.

To code ω-words by sets of natural numbers, we assume without loss of generality that $\Sigma = \{0,1\}^k$. With this convention a word $\alpha \in \Sigma^\omega$ consists of k words $\alpha_1, \ldots, \alpha_k \in \{0,1\}^\omega$. We code α_i with the set X_i, where $x \in X_i$ iff $\alpha_i(x) = 1$. Then the tuple $\underline{\alpha} = (X_1, \ldots, X_k)$ is a unique coding of α. We will refer to natural numbers as "positions". Now we can express the fact that a word has a certain letter $a = (a_1, \ldots, a_k)$ at the position x by the formula $\mathrm{POS}_a(x, X_1, \ldots, X_k) = \bigwedge_{i \in O(a)} X_i x \wedge \bigwedge_{i \in Z(a)} \neg X_i x$, where $O(a) = \{i | a_i = 1\}$ and $Z(a) = \{i | a_i = 0\}$.

Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be an alternating weak parity automaton with $c : Q \rightarrow \{0, \ldots, k\}$ and without $\top$ and $\bot$ in the transition function (this can be obtained by adding at most one extra state). We have to code the runs of an automaton on a word with subsets of $\mathbb{N}$. Let $Q = \{1, \ldots, n\}$ with $q_0 = 1$ and let $m = n + n^2$. We code a level l of a run and the edges to the previous level with a vector $v_l \in \{0,1\}^m$. The first n entries code the active states. This means entry i is 1 iff (i, l) is a vertex of the run. For every $i \in \{1, \ldots n\}$ the entries from $i \cdot n + 1$ to $i \cdot n + n$ code the successors of the vertex $(i, l-1)$. This means entry $i \cdot n + j$ is 1 iff (j, l) is a successor of $(i, l-1)$. This idea is illustrated in Figure 2 for the beginning of a run of an automaton with states $\{q_0, q_1, q_2, q_3\}$.

This coding yields an infinite sequence $v_0, v_1, \ldots$ of vectors from $\{0,1\}^m$ which can also be represented by $Y_1, \ldots, Y_m \subseteq \mathbb{N}$ in the same way as the words

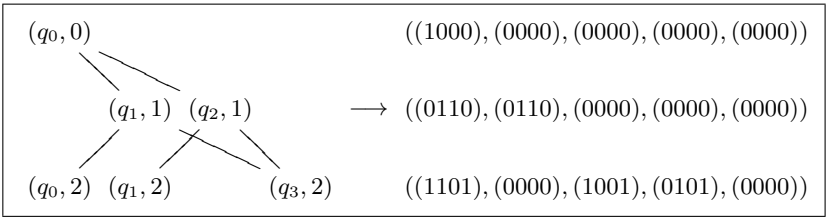


Fig. 2. Coding of the beginning of a run

from Σ^ω. It is easy to verify that the first-order formula

$$\begin{aligned}
&\mathrm{RUN}_{\mathcal{A}}(X_1,\dots,X_k,Y_1,\dots,Y_m) = (Y_1 0 \wedge \bigwedge_{i=2}^{m} \neg Y_i 0)\\
&\wedge \forall x \Bigg[\bigwedge_{i=1}^{n} (x>0 \wedge Y_i x \to (\bigvee_{j=1}^{n} Y_{j\cdot n+i} x)) \ \wedge \bigwedge_{i=1}^{n} (\neg Y_i x \to (\bigwedge_{j=1}^{n} \neg Y_{i\cdot n+j} x + 1))\\
&\wedge \bigwedge_{(i,a)\in Q\times\Sigma} (Y_i x \wedge \mathrm{POS}_a(x, X_1,\dots,X_k)\\
&\qquad\qquad \to \bigvee_{S\in\mathcal{M}_{\delta(i,a)}} (\bigwedge_{j\in S} Y_j x + 1 \bigwedge_{j\in S} Y_{i\cdot n+j} x + 1 \wedge \bigwedge_{j\notin S} \neg Y_{i\cdot n+j} x + 1))\Bigg]
\end{aligned}$$

is satisfied iff $Y_1,\dots,Y_m$ code a run of $\mathcal{A}$ on the input coded by $X_1,\dots,X_k$. In the same way we can construct a formula $\mathrm{DUALRUN}_{\mathcal{A}}(X_1,\dots,X_k,Y_1,\dots,Y_m)$ that defines the runs of the dual automaton of $\mathcal{A}$.

To express the acceptance of a word by an automaton we now have to code the paths of a run. A path through a run $G = (V,E)$ can be viewed as a subgraph $G' = (V',E')$ of G ($V' \subseteq V$ and $E' \subseteq E$) such that every vertex has exactly one successor and every vertex except the initial vertex has exactly one predecessor. We can code a path by $Z_1,\dots Z_n \subseteq \mathbb{N}$. These Z_i must have the following properties (with i and j ranging over $\{1,\dots,n\}$):

- $Z_i \subseteq Y_i$ for every i ($V' \subseteq V$),
- $Z_i \cap Z_j = \emptyset$ for every $i \neq j$ (in every level is at most one vertex),
- $\forall x \exists i (x \in Z_i)$ (in every level is at least one vertex),
- for all x with $Z_i x$ and $Z_j x+1$ one has $Y_{i\cdot n+j} x+1$ (the vertex in level $x+1$ is a successor of the vertex in level x),

and we indicate by $\mathrm{PATH}_{\mathcal{A}}(Y_1,\dots,Y_m,Z_1,\dots,Z_n)$ a first-order formula expressing this.

The last fact we have to express is that a path satisfies the acceptance condition. We define for $m \in \{0,\dots,k\}$ the set Q_m of states with color m, i.e., $Q_m = \{q \in Q \mid c(q) = m\}$. Let $Z_1,\dots,Z_n \subseteq \mathbb{N}$ be the coding of a path through a run. Then the first-order formula

$$\mathrm{WEAKACC}_{\mathcal{A}}(Z_1,\dots,Z_n) = \bigvee_{m=0}^{k} \Big(\exists x (\bigvee_{i\in Q_m} Z_i x) \wedge \forall x (\bigwedge_{i\in \bigcup_{l<m} Q_l} \neg Z_i x)\Big)$$

is satisfied if and only if the path coded by $Z_1, \ldots, Z_n$ satisfies the acceptance condition of $\mathcal{A}$.

Now we can translate an automaton into two equivalent S1S formulas. Here we write $\overline{X}$ for $X_1, \ldots, X_k$ (and similar $\overline{Y}$ and $\overline{Z}$).

Theorem 3. *The following two formulas are satisfied if and only if $\overline{X}$ is the coding of a word $\alpha \in L(\mathcal{A})$.*

$$\phi_{\mathcal{A}}(\overline{X}) = \exists\overline{Y}\forall\overline{Z}(\mathrm{RUN}_{\mathcal{A}}(\overline{X},\overline{Y}) \wedge\ (\mathrm{PATH}_{\mathcal{A}}(\overline{Y},\overline{Z}) \rightarrow\ \mathrm{WEAKACC}_{\mathcal{A}}(\overline{Z}))).$$

$$\widetilde{\phi}_{\mathcal{A}}(\overline{X}) = \forall\overline{Y}\exists\overline{Z}(\mathrm{DUALRUN}_{\mathcal{A}}(\overline{X},\overline{Y}) \rightarrow\ (\mathrm{PATH}_{\mathcal{A}}(\overline{Y},\overline{Z}) \wedge\ \mathrm{WEAKACC}_{\mathcal{A}}(\overline{Z}))).$$

The formulas $\phi_{\mathcal{A}}$ and $\widetilde{\phi}_{\mathcal{A}}$ thus represent a normal form for S1S-formulas of second-order quantifier prefix types Σ_2, Π_2, respectively, in which the acceptance component WEAKACC only involves reachability conditions.

6 The Landweber Hierarchy

The classical characterization of regular sequence properties in the Landweber hierarchy [Lan69] uses deterministic automata with different acceptance conditions. As we show, the hierarchy can also be characterized in two different ways: first by alternating automata, equipped with weak acceptance conditions, and second by fixing the acceptance condition as weak parity, and modifying the mode of the transition function. The three different characterizations are shown in Figure 3. For notational simplicity we abbreviate the type of an automaton by the initial letters of its transition mode and its type of acceptance condition. So for example UWCB denotes universal weak co-Büchi automata and DP denotes deterministic parity automata. If T is such an identifier, then $\mathcal{L}(T)$ denotes the class of languages characterized by automata of type T. As explained in [MP92], the language classes $\mathcal{L}$(DB), $\mathcal{L}$(DCB), and $\mathcal{L}$(DB)$\cap\mathcal{L}$(DCB) capture the (regular) "recurrence properties", "persistence properties", and "obligation properties", respectively. So our results clarify their role in the framework of alternating automata.

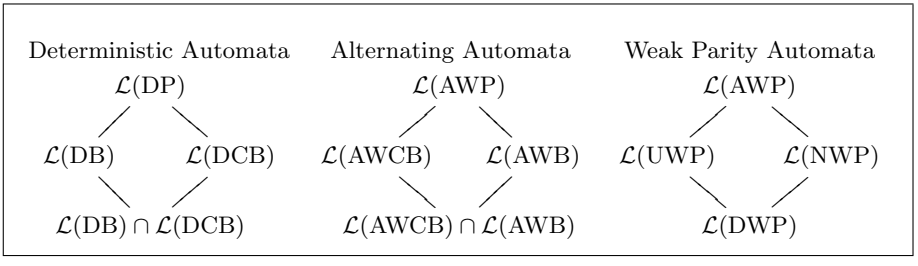


Fig. 3. Three different characterizations of the Landweber hierarchy

Before we give the theorem stating the correctness of the characterization from Figure 3, we need some preparations.

Lemma 4. *For every deterministic Büchi automaton one can construct an equivalent universal weak co-Büchi automaton.*

Proof. For the transformation we use a simplified version of the construction from Section 4.

In [MH84] Miyano and Hayashi give an exponential transformation of alternating Büchi automata into nondeterministic Büchi automata. But the construction also gives a more general theorem, which is stated below.

Theorem 4. *Let $\mathcal{A}$ be an alternating Büchi (weak Büchi) automaton. One can construct an equivalent nondeterministic Büchi (weak Büchi) automaton $\mathcal{A}'$ and furthermore, if $\mathcal{A}$ is universal, then $\mathcal{A}'$ is deterministic.*

To apply this theorem to weak parity automata, we give a transformation of weak parity automata into Büchi automata. The idea is to remember the lowest color seen so far. This suffices to decide with a Büchi condition, whether a path is accepting or not.

Lemma 5. *Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ be a weak parity automaton with $c : Q \to C$. One can construct an equivalent Büchi automaton $\mathcal{A}' = (Q', \Sigma, q_0', \delta', F')$ with the same mode of transition function as $\mathcal{A}$.*

Theorem 5. *(1)* $\mathcal{L}(\mathrm{DP}) = \mathcal{L}(\mathrm{AWP})$.
(2) $\mathcal{L}(\mathrm{DB}) = \mathcal{L}(\mathrm{AWCB}) = \mathcal{L}(\mathrm{UWP})$.
(3) $\mathcal{L}(\mathrm{DCB}) = \mathcal{L}(\mathrm{AWB}) = \mathcal{L}(\mathrm{NWP})$.
(4) $\mathcal{L}(\mathrm{DB}) \cap \mathcal{L}(\mathrm{DCB}) = \mathcal{L}(\mathrm{AWB}) \cap \mathcal{L}(\mathrm{AWCB}) = \mathcal{L}(\mathrm{DWP})$.

Proof. (1): This follows from Theorem 2 and from the fact that every language recognized by an alternating weak parity automaton is regular.

(2): From Lemma 4 follows $\mathcal{L}(\mathrm{DB}) \subseteq \mathcal{L}(\mathrm{AWCB})$ and $\mathcal{L}(\mathrm{DB}) \subseteq \mathcal{L}(\mathrm{UWP})$. Since one can transform every universal weak parity automaton into a universal Büchi automaton by Lemma 5, and then into a deterministic Büchi automaton by Theorem 4, we get $\mathcal{L}(\mathrm{UWP}) \subseteq \mathcal{L}(\mathrm{DB})$.

If we are given an alternating weak co-Büchi automaton, we dualize it, yielding an alternating weak Büchi automaton. This can be transformed into a nondeterministic weak Büchi automaton by Theorem 4. If we dualize again, we get a universal weak co-Büchi automaton, equivalent to the given automaton. Therefore we get $\mathcal{L}(\mathrm{AWCB}) \subseteq \mathcal{L}(\mathrm{UWP})$, since weak co-Büchi conditions are special cases of weak parity conditions.

We omit the proof of (3) because its the dual statement to (2), and (4) can be shown very easily, using (2) and (3), and some criteria on the loop structure of deterministic ω-automata [Lan69].

7 The PLTL Fragment

In the previous section we gave exact characterizations for the different fragments from the Landweber hierarchy. Another fragment of the regular ω-languages is

the fragment of PLTL (propositional linear temporal logic, see [Eme90]) that includes all languages that can be described by PLTL formulas, or, equivalently, by first-order formulas. PLTL formulas are built up from a finite set P of atomic propositions, the boolean operators, and the temporal operators $\bigcirc$ (Next), $\Box$ (Always), $\Diamond$ (Eventually), $\mathcal{U}$ (Until). For this section we fix the alphabet $\Sigma = 2^P$. Let $\alpha \in \Sigma^\omega$, $i \in \mathbb{N}$, $p \in P$, and φ, φ' be PLTL formulas. The relation $\models$ is defined as follows.

$$\begin{array}{ll}
\alpha, i \models p & \text{if } p \in \alpha(i), \\
\alpha, i \models \bigcirc\varphi & \text{if } \alpha, i+1 \models \varphi, \\
\alpha, i \models \Box\varphi & \text{if } \forall k \geq i\ (\alpha, k \models \varphi), \\
\alpha, i \models \Diamond\varphi & \text{if } \exists k \geq i\ (\alpha, k \models \varphi), \\
\alpha, i \models \varphi \mathrm{U} \varphi' & \text{if } \exists k \geq i\ (\alpha, k \models \varphi' \text{ and } \forall i \leq j < k\ (\alpha, j \models \varphi)).
\end{array}$$

For the boolean operators, $\models$ is defined in the straightforward way.

A PLTL formula φ defines the language $L(\varphi) = \{\alpha \in \Sigma^\omega \mid \alpha, 0 \models \varphi\}$. The PLTL fragment of the regular ω-languages is the class of all languages that can be defined by a PLTL formula.

In this section we give an exact automata theoretic characterization of this fragment in terms of a subclass of alternating weak parity automata, so called alternating linear automata.

An alternating weak parity automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, c)$ is called a *linear automaton*, if in the transition graph there are no cycles containing 2 or more states, and if along each path through the transition graph the colors of the states do not increase.

A simple induction shows that a PLTL-definable language can also be recognized by an alternating linear automaton (see e.g. [Var97]). Here we show the other direction, which was independently shown by Rohde [Roh97].

Theorem 6. *Let $\mathcal{A} = (Q, \Sigma, \theta_0, \delta, c)$ be an alternating linear automaton. There is a PLTL formula χ such that $L(\mathcal{A}) = L(\chi)$.*

Proof. For $\theta \in \mathcal{B}^+(Q)$ let $\mathcal{A}(\theta) = (Q, \Sigma, \theta, \delta, c)$ (an automaton with an initial formula instead of an initial state). For $R \subseteq P$ let $\psi_R = \bigwedge_{p \in R} p \wedge \bigwedge_{p \notin R} \neg p$.

We construct for every $\theta \in \mathcal{B}^+(Q)$ a PLTL formula $\chi(\theta)$ with $L(\chi(\theta)) = L(\mathcal{A}(\theta))$ and then set $\chi = \chi(\theta_0)$. Let $\theta \in \mathcal{B}^+(Q)$. If for all $q \in Q$ that occur in θ the formula $\chi(q)$ is already known, then we obtain $\chi(\theta)$ by replacing each atom q in θ by the formula $\chi(q)$. Furthermore we set $\chi(\top) = \top$ and $\chi(\bot) = \bot$.

Let $q \in Q$ and let $Tr(q)$ denote the set of states q' such that a transition leads from q to q'. Since $\mathcal{A}$ is a linear automaton, we can assume by induction that for all $q' \in Tr(q) \setminus \{q\}$ the formula $\chi(q')$ is already known, and therefore also all $\chi(\theta)$ with $\theta \in \mathcal{B}^+(Tr(q) \setminus \{q\})$ are known.

For all $R \subseteq P$ we can write the transition formula for q and R in the form $\delta(q, R) = (q \wedge \theta_{R,q}) \vee \theta'_{R,q}$ with $\theta_{R,q}, \theta'_{R,q} \in \mathcal{B}^+(Tr(q) \setminus \{q\})$. We define

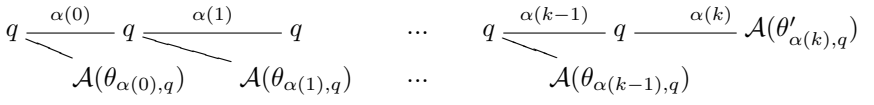
$$\chi(q) = \begin{cases} \varphi_q \mathcal{U} \varphi'_q & \text{if } c(q) \text{ is even,} \\ \varphi_q \mathcal{U} \varphi'_q \vee \Box\varphi_q & \text{if } c(q) \text{ is odd,} \end{cases}$$

with

$$\varphi_q = \bigvee_{R \subseteq P} (\psi_R \wedge \bigcirc \chi(\theta_{R,q})) \text{ and } \varphi'_q = \bigvee_{R \subseteq P} (\psi_R \wedge \bigcirc \chi(\theta'_{R,q})).$$

To show that for all $\theta \in \mathcal{B}^+(Q)$ the equality $L(\chi(\theta)) = L(\mathcal{A}(\theta))$ holds, it suffices to show $L(\chi(q)) = L(\mathcal{A}(q))$ for all $q \in Q$.

Let $q \in Q$ and let $\alpha \in L(\chi(q))$. If $c(q)$ is odd, then there exists a $k \in \mathbb{N}$ such that $\alpha, i \models \varphi_q$ for all $i < k$ and $\alpha, k \models \varphi'_q$. Thus, for all $i < k$, the word $\alpha[i+1, \infty)$ is in $L(\chi(\theta_{\alpha(i),q}))$ and $\alpha[k+1, \infty)$ is in $L(\chi(\theta'_{\alpha(k),q}))$. By induction we know that $L(\chi(\theta_{\alpha(i),q})) = L(\mathcal{A}(\theta_{\alpha(i),q})$ and $L(\chi(\theta'_{\alpha(k),q})) = L(\mathcal{A}(\theta'_{\alpha(k),q})$. An accepting run of $\mathcal{A}(q)$ has the following form:



The identifiers of the automata stand for accepting runs of these automata on the corresponding suffix of α.

If $c(q)$ is even then the proof is analogous accept for the case that the $\Box\varphi_q$ part is satisfied. Then we get a run of $\mathcal{A}_q$ with an infinite path labelled with q. This run is also accepting, because $c(q)$ is even.

For the other direction let $\alpha \in L(\mathcal{A}_q)$. If $c(q)$ is odd, then an accepting run of $\mathcal{A}_q$ on α is of the form as given above. But then, using the induction hypothesis, $\alpha \models \varphi_q \mathcal{U} \varphi'_q$. In case $c(q)$ is even, we can also get an accepting run with an infinite path labelled with q, but then $\alpha \models \Box\varphi_q$.

Acknowledgement

We thank Olivier Carton, Marcin Jurdziński, Narayan Kumar, Madhusudan, and Madhavan Mukund, as well as the referees for useful comments and discussions.

References

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. International Congress on Logic, Method and Philos. Sci. 1960*, pages 1–11, 1962.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. v. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.

[Kla91] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd FOCS*, pages 358–367, 1991.

[KV97] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symposium on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[Lan69] L.H. Landweber. Decision poblems for ω-automata. *Math. System Theory*, 3:376–384, 1969.

[Löd98] C. Löding. Methods for the transformation of ω-automata: Complexity and connection to second order logic. Master's thesis, Christian-Albrechts-University of Kiel, 1998.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[MSS86] D.E. Muller, A. Saoudi, and P.E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th ICALP, LNCS 226*, pages 275–283, 1986.

[Roh97] S. Rohde. *Alternating automata and the temporal logic of ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[Tho97] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 385–455. Springer-Verlag, 1997.

[Tho99] W. Thomas. Complementation of Büchi automata revisited. In J. Karhumäki et al., editor, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122. Springer, 1999.

[Var97] M.Y. Vardi. Alternating automata: Unifying truth and validity checking for temporal logics. In *14th International Conference on Automated Deduction, Lecture Notes in Artificial Intelligence 1249*, pages 191–206, 1997.

Hypothesis Support for Information Integration in Four-Valued Logics

Yann Loyer, Nicolas Spyratos*, and Daniel Stamate

Laboratoire de Recherche en Informatique, UMR 8623,
Université de Paris Sud, Bat. 490, 91405 Orsay
{loyer,spyratos,daniel}@lri.fr

Abstract. We address the problem of integrating information coming from different sources. The information consists of facts that a central server collects and tries to combine using (a) a set of logical rules, i.e. a logic program, and (b) a hypothesis representing the server's own estimates. In such a setting incomplete information from a source or contradictory information from different sources necessitate the use of many-valued logics in which programs can be evaluated and hypotheses can be tested. To carry out such activities we propose a formal framework based on Belnap's four-valued logic. In this framework we work with the class of programs defined by Fitting and we develop a theory for information integration. We also establish an intuitively appealing connection between our hypothesis testing mechanism on the one hand, and the well-founded semantics and Kripke-Kleene semantics of Datalog programs with negation, on the other hand.

Keywords : deductive databases and knowledge bases, information integration, logics of knowledge, inconsistency, four-valued logic.

1 Introduction

In several information oriented activities there is a need for combining (or "integrating") information coming from different sources.

A typical example of such information-oriented activity is building a data warehouse, i.e. a special kind of very large database for decision-making support in big enterprises [1]. The information stored in a data warehouse is obtained from queries to operational databases internal to the enterprise, and from remote information sources external to the enterprise accessed through the Internet. The answers to all such queries are then combined (by the so-called "integrator") to derive the information to be stored in the data warehouse.

The basic pattern of the data warehouse paradigm, i.e. collection of information then integration, is encountered in many different situations. What changes usually from one situation to another is the type (and volume) of the collected

* Part of this research work was conducted while this author was visiting with the Meme Media Laboratory, University of Hokkaido, Japan.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 536–548, 2000.
© Springer-Verlag Berlin Heidelberg 2000

information and the means used for the integration.

In this paper we address a specific problem of information integration, namely, the information consists of facts that a central server collects from a number of autonomous sources and then tries to combine them using:

- a set of logical rules, i.e. a logic program, and
- a hypothesis, representing the server's own estimates.

In such a setting incomplete information from a source or contradictory information coming from different sources necessitate the use of many-valued logics, in which programs can be evaluated and hypotheses can be tested. Let us see a simple example.

Example 1. Consider a legal case where a judge (the "central server") has to decide whether to charge a person named John accused of murder. To do so, the judge first collects facts from two different sources: the public prosecutor and the person's lawyer. The judge then combines the collected facts using a set of rules in order to reach a decision. For the sake of our example let us suppose that the judge has collected a set of facts F that he combines using a set of rules R as follows:

$$F \quad \begin{bmatrix} \text{witness(John)} & \text{friends(John, Ted)} \\ \downarrow & \downarrow \\ \text{false} & \text{true} \end{bmatrix}$$

$$R \quad \begin{cases} \text{suspect(X)} & \leftarrow \text{motive}(X) \vee \text{witness}(X) \\ \text{innocent}(X) & \leftarrow \exists Y(\text{alibi}(X,Y) \wedge \neg\text{friends}(X,Y)) \\ \text{friends}(X,Y) & \leftarrow \text{friends}(Y,X) \vee (\text{friends}(X,Z) \wedge \text{friends}(Z,Y)) \\ \text{charge}(X) & \leftarrow \text{suspect}(X) \oplus \neg\text{innocent}(X) \end{cases}$$

The first fact of F says that there is no witness, i.e. the fact witness($John$) is false. The second fact of F says that Ted is a friend of John, i.e. the fact friends($John, Ted$) is true.

Turning now to the set of rules, the first rule of R describes how the prosecutor works: in order to support the claim that a person X is a suspect, the prosecutor tries to provide a motive and/or a witness.

The second rule of R describes how the lawyer works: in order to support the claim that X is innocent, the lawyer tries to provide an alibi for X by a person who is not a friend of X. This rule depends on the third rule which defines the relation $friends$.

Finally, the fourth rule of R is the "decision making rule" and describes how the judge works: in order to reach a decision as to whether to charge X, the judge examines the premises $suspect(X)$ and $\neg innocent(X)$. As explained earlier, the values of these premises come from two different sources: the prosecutor and the lawyer. Each of these premises can have the value true or false. However, it is also possible that the value of a premiss is undefined. For example, if a motive is

not known and a witness has not been found, then the value of suspect(X) will be undefined.

In view of these observations, the question is what value is appropriate to associate with charge(X).

What we propose is to collect together the values of the premises suspect(X) and ¬innocent(X), and to consider the resulting set of values as the value of charge(X). This is precisely what the notation

$$\text{charge}(X) \leftarrow \text{suspect}(X) \oplus \neg\text{innocent}(X)$$

means, where $\oplus$ denotes the "collecting together" operation.

It follows that there are four possible values for charge(X) : $\emptyset$, {true}, {false} and {true, false}. We shall call these values : $Underdefined$, $True$, $False$ and $Overdefined$, and we shall denote them by $\mathcal{U}$, $\mathcal{T}$, $\mathcal{F}$ and $\mathcal{O}$, respectively.

The value $Underdefined$ for a premiss means that the premiss is true or false but its actual value is currently unknown. For the purpose of this paper we shall assume that any premiss whose value is not known is associated with the value $Underdefined$.

We note that the value $Underdefined$ is related to the so-called "null values" of attributes in database theory. In database theory, however, a distinction is made between two types of null values [13]:

- the attribute value exists but is currently unknown
- the attribute value does not exist

An example of the first type is the Department-value for an employee that has just been hired but has not yet been assigned to a specific department, and an example of the second type is the maiden name of a male employee. The value $Underdefined$ corresponds to the first type of null value.

Returning now to our example, the decision whether to charge John depends on the value that charge(John) will receive when collecting the values of the premises together. Looking at the facts of F and the rules of R (and using intuition) we can see that suspect(John) and innocent(John) both receive the value $\mathcal{U}$ and so then does charge(John).

This is clearly a case where the judge cannot decide whether to actually charge John!

In the context of decision making, however, one has to reach a decision (based on the available facts and rules) even if some values are not defined. This can be accomplished by *assuming* values for some or all underdefined premises. Such an assignment of values to underdefined premises is what we call a *hypothesis*.

Thus in our example, if the judge assumes the innocence of John, then charge(John) receives the value false and John is not charged. We note that this is precisely what happens in real life under similar circumstances, i.e. the defendant is *assumed* innocent until proved guilty.

Clearly, when hypothesizing on underdefined premises we would like our hypothesis to be "reasonable" in some sense, with respect to the available informa-

tion, i.e., with respect to the given facts and rules. Roughly speaking, we define a hypothesis H to be "reasonable" or *sound* using the following test :

calling a fact f *defined* under H if $H(f) \neq \mathcal{U}$,

1. add H to F to produce a new set of facts $F' = F \cup H$;
2. apply the rules of R to F' to produce a new assignment of values H';
3. if the facts defined under H are assigned to the same values in H' then H is sound, otherwise H is not sound.

That is, if there is no fact of H that has changed value as a result of rule application then H is a sound hypothesis; otherwise H is unsound.

In our example, for instance, consider the following hypothesis:

$$H_1 = \begin{bmatrix} \text{innocent(John)} & \text{charge(John)} \\ \downarrow & \downarrow \\ \mathcal{T} & \mathcal{T} \end{bmatrix}$$

Applying the above test we find the following values for the facts of H_1 :

$$H_1' = \begin{bmatrix} \text{innocent(John)} & \text{charge(John)} \\ \downarrow & \downarrow \\ \mathcal{T} & \mathcal{F} \end{bmatrix}$$

As we can see, the fact charge(John) has changed value, i.e. this fact had the value $\mathcal{T}$ in H_1 and now has the value $\mathcal{F}$ in H_1'. Therefore, H_1 is not a sound hypothesis.

Next, consider the following hypothesis:

$$H_2 = \begin{bmatrix} \text{innocent(John)} & \text{charge(John)} \\ \downarrow & \downarrow \\ \mathcal{T} & \mathcal{F} \end{bmatrix}$$

Applying again our test we find :

$$H_2' = \begin{bmatrix} \text{innocent(John)} & \text{charge(John)} \\ \downarrow & \downarrow \\ \mathcal{T} & \mathcal{F} \end{bmatrix}$$

That is, the values of the facts of H_2 remain unchanged in H_2', thus H_2 is a sound hypothesis.

Intuitively, if our hypothesis is sound this means that what we have assumed is compatible with the given facts and rules.

From now on let us denote $\mathcal{P}$ the facts of F together with the rules of R, i.e. $\mathcal{P} = \langle F, R \rangle$, and let us call $\mathcal{P}$ a program.

In principle, we may assume or hypothesize values for every possible ground atom. However, given a program $\mathcal{P}$ and a hypothesis H, we cannot expect H to be sound with respect to $\mathcal{P}$, in general. What we can expect is that some "part" of H is sound with respect to $\mathcal{P}$.

More precisely, given two hypotheses H and H', call H a *part of* H', denoted $H \leq H'$, if $H(f) \neq \mathcal{U}$ implies $H(f) = H'(f)$, i.e., if H agrees with H' on every

defined fact. It is then natural to ask, given program $\mathcal{P}$ and hypothesis H, what is the maximal part of H that is sound with respect to $\mathcal{P}$. We call this maximal part the *support* of H by $\mathcal{P}$, and we denote it by $s^H_{\mathcal{P}}$. Intuitively, the support of H indicates how much of H can be assumed safely, i.e., remaining compatible with the facts and rules of $\mathcal{P}$.

We show that the support $s^H_{\mathcal{P}}$ can be used to define a hypothesis-based semantics of $\mathcal{P} = \langle F, R\rangle$, denoted by $sem^H_{\mathcal{P}}$. This is done by a fixpoint computation that uses an immediate consequence operator T as follows:

- $F_0 = F$;
- $F_{i+1} = T(F_i) \oplus s^H_{\langle F_i, R\rangle}$.

We also show that there is an interesting connection between hypothesis based semantics and the semantics of Datalog programs with negation. More precisely, we show that if $\mathcal{P}$ is a Datalog program with negation then:

- if H is the everywhere false hypothesis then $sem^H_{\mathcal{P}}$ coincides with the well-founded semantics of $\mathcal{P}$ [11,12], and
- if H is the everywhere underdefined hypothesis then $sem^H_{\mathcal{P}}$ coincides with the Kripke-Kleene semantics of $\mathcal{P}$ [4].

As we shall see, these results allow us to extend the well-founded semantics and the Kripke-Kleene semantics of Datalog program with negation to the broader class of Fitting programs [6].

Motivation for this work comes from the area of knowledge acquisition, where contradictions may occur during the process of collecting knowledge from different experts. Indeed, in multi-agent systems, different agents may give different answers to the same query. It is then important to be able to process the answers so as to extract the maximum of information on which the various agents agree, or to detect the items on which the agents give conflicting answers.

Motivation also comes from the area of deductive databases. Updates leading to a certain degree of inconsistency should be allowed because inconsistency can lead to useful information, especially within the framework of distributed databases. In particular, Fuhr and Rölleke showed in [7] that hypermedia retrieval requires the handling of inconsistent information.

The remaining of the paper is organized as follows. In Section 2 we recall very briefly some definitions and notations from well-founded semantics, Belnap's logic $\mathcal{FOUR}$ and Fitting programs. We then proceed, in Section 3, to define sound hypotheses and their support by a Fitting program $\mathcal{P}$; we also discuss computational issues and we present algorithms for computing the support of a hypothesis by a program $\mathcal{P}$ and the hypothesis-based semantics of $\mathcal{P}$. In Section 4 we show that the notion of support actually unifies the notions of well-founded semantics and Kripke-Kleene semantics and extends them from Datalog program with negation to the broader class of Fitting programs. Section 5 contains concluding remarks and suggestions for further research. Proofs of theorems are omitted due to lack of space.

2 Preliminaries

2.1 Three-Valued logics

Well founded semantics Well-founded semantics of logic programs were first proposed in [11]. In the approach of [11] an interpretation I is a set of ground literals that does not contain literals of the form A and $\neg A$. Now, if we consider an instantiated program P defined as in [11], its well-founded semantics is defined using the following two operators on partial interpretations I :

- the immediate consequence operator T_P, defined by

$$T_P(I) = \{head(r) \mid r \in P \wedge \forall B \in body(r), B \in I\}, \text{ and}$$

- the unfounded operator U_P, where $U_P(I)$ is defined to be the greatest unfounded set with respect to the partial interpretation I.

We recall that a set of instantiated atoms U is said to be unfounded with respect to I if for all instantiated atoms $A \in U$ and for all rules $r \in P$ the following holds:

$$head(r) = A \Rightarrow \exists B \in body(r)\ (\neg B \in I \vee B \in U)$$

In [3] it is proven that $U_P(I) = \mathcal{HB} \setminus SPF_P(I)$, where $\mathcal{HB}$ is the Herbrand Base and $SPF_P(I)$ is the limit of the increasing sequence $[SPF^i(I)]_{i\geq 1}$ defined by:

- $SPF_P^1(I) = \{head(r) \mid r \in P \wedge pos(body(r)) = \emptyset \wedge \forall B \in body(r), \neg B \notin I\}$
- $SPF_P^{i+1}(I) = \{head(r) \mid r \in P \wedge pos(body(r)) \subseteq SPF_P^i(I) \wedge \forall B \in body(r), \neg B \notin I\}, i > 0.$

The atoms of $SPF_P(I)$ are called potentially founded atoms.

The operator W_P, called the well-founded operator, is then defined by $W_P(I) = T_P(I) \cup \neg U_P(I)$ and is shown to be monotone with respect to set inclusion. The well-founded semantics of P is defined to be the least fixpoint of W_P [11].

Kripke-Kleene semantics The Kripke-Kleene semantics was introduced in [4]. In the approach of [4], a valuation is a function from the Herbrand base to the set of logical values $\{true,\ false,\ unknown\}$. Now, given an instantiated program $\mathcal{P}$ defined as in [4], its Kripke-Kleene semantics is defined using an operator Φ on valuations, defined as follows : given a ground atom A,

- if there is a rule in $\mathcal{P}$ with head A, and the truth value of the body under v is $true$, then $\Phi_{\mathcal{P}}(v)(A) = true$;
- if there is a rule in $\mathcal{P}$ with head A, and for every rule in $\mathcal{P}$ with head A the truth value of the body under v is false, then $\Phi_{\mathcal{P}}(v)(A) = false$;
- else $\Phi_{\mathcal{P}}(v)(A) = unknown$.

2.2 Four-Valued Logics

Belnap's four-valued logic In [2], Belnap defines a logic called $\mathcal{FOUR}$ intended to deal with incomplete and inconsistent information. Belnap's logic uses four logical values that we shall denote by $\mathcal{F}$, $\mathcal{T}$, $\mathcal{U}$ and $\mathcal{O}$, i.e. $\mathcal{FOUR} = \{\mathcal{F}, \mathcal{T}, \mathcal{U}, \mathcal{O}\}$. These values can be compared using two orderings, the knowledge ordering and the truth ordering.

In the knowledge ordering, denoted by $\leq_k$, the four values are ordered as follows: $\mathcal{U} \leq_k \mathcal{F}$, $\mathcal{U} \leq_k \mathcal{T}$, $\mathcal{F} \leq_k \mathcal{O}$, $\mathcal{T} \leq_k \mathcal{O}$. Intuitively, according to this ordering, each value of $\mathcal{FOUR}$ is seen as a possible knowledge that one can have about the truth of a given statement. More precisely, this knowledge is expressed as a set of classical truth values that hold for that statement. Thus, $\mathcal{F}$ is seen as $\{false\}$, $\mathcal{T}$ is seen as $\{true\}$, $\mathcal{U}$ is seen as $\emptyset$ and $\mathcal{O}$ is seen as $\{false,true\}$. Following this viewpoint, the knowledge ordering is just the set inclusion ordering.

In the truth ordering, denoted by $\leq_t$, the four logical values are ordered as follows: $\mathcal{F} \leq_t \mathcal{U}, \mathcal{F} \leq_t \mathcal{O}, \mathcal{U} \leq_t \mathcal{T}, \mathcal{O} \leq_t \mathcal{T}$. Intuitively, according to this ordering, each value of $\mathcal{FOUR}$ is seen as the degree of truth of a given statement. $\mathcal{U}$ and $\mathcal{O}$ are both less false than $\mathcal{F}$, and less true than $\mathcal{T}$, but $\mathcal{U}$ and $\mathcal{O}$ are not comparable.

The two orderings are represented in the double Hasse diagram of Figure 1.

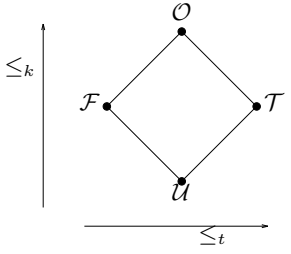


Fig. 1. The logic *FOUR*

Both $\leq_t$ and $\leq_k$ give $\mathcal{FOUR}$ a lattice structure. Meet and join under the truth ordering are denoted by $\wedge$ and $\vee$, and they are natural generalizations of the usual notions of conjunction and disjunction. In particular, $\mathcal{U} \wedge \mathcal{O} = \mathcal{F}$ and $\mathcal{U} \vee \mathcal{O} = \mathcal{T}$. Under the knowledge ordering, meet and join are denoted by $\otimes$ and $\oplus$, and are called the *consensus* and *gullibility*, respectively: $x \otimes y$ represents the maximal information on which x and y agree, whereas $x \oplus y$ adds the knowledge represented by x to that represented by y. In particular, $\mathcal{F} \otimes \mathcal{T} = \mathcal{U}$ and $\mathcal{F} \oplus \mathcal{T} = \mathcal{O}$.

There is a natural notion of negation in the truth ordering denoted by $\neg$, and we have: $\neg\, \mathcal{T} = \mathcal{F}$, $\neg\, \mathcal{F} = \mathcal{T}$, $\neg\, \mathcal{U} = \mathcal{U}$, $\neg\, \mathcal{O} = \mathcal{O}$. There is a similar notion for the knowledge ordering, called *conflation*, denoted by -, and we have: $-\, \mathcal{U} = \mathcal{O}$, $-\, \mathcal{O} = \mathcal{U}$, $-\, \mathcal{F} = \mathcal{F}$, $-\, \mathcal{T} = \mathcal{T}$.

The operations $\vee, \wedge, \neg$ restricted to the values $\mathcal{T}$ and $\mathcal{F}$ are those of classical logic, and if we add to these operations and values the value $\mathcal{U}$then they are those of Kleene's strong three-valued logic.

Fitting programs Conventional logic programming has the set $\{\mathcal{F}, \mathcal{T}\}$ as its intended space of truth values but since not every query may produce an answer partial models are often allowed (i.e. $\mathcal{U}$ is added). If we want to deal with inconsistency as well then $\mathcal{O}$ must be added. Thus Fitting asserts that $\mathcal{FOUR}$ can be thought as the "home" of ordinary logic programming and extends the notion of logic program, as follows:

Definition 1. (Fitting program)

- *A formula is an expression built up from literals and elements of* $\mathcal{FOUR}$*, using* $\wedge, \vee, \otimes, \oplus, \exists, \forall$*.*
- *A clause is of the form* $P(x_1, ..., x_n) \longleftarrow \phi(x_1, ..., x_n)$*, where the atomic formula* $P(x_1, ..., x_n)$ *is the head, and the formula* $\phi(x_1, ..., x_n)$ *is the body. It is assumed that the free variables of the body are among* $x_1, ..., x_n$*.*
- *A program is a finite set of clauses with no predicate letter appearing in the head of more than one clause (this apparent restriction causes no loss of generality [5]).*

We shall represent a *Fitting program* as a pair $\langle F, R\rangle$ where F is a function from the Herbrand base into $\mathcal{FOUR}$ and R a set of clauses. This is possible because every fact can be seen as a rule of the form $A \leftarrow v$, where A is an atom and v is a value in $\mathcal{FOUR}$.

A Datalog program with negation can be seen as a Fitting program whose underlying truth-value space is the subset $\{\mathcal{F}, \mathcal{T}, \mathcal{U}\}$ of $\mathcal{FOUR}$ and which does not involve $\otimes, \oplus, \forall, \mathcal{U}, \mathcal{O}, \mathcal{F}$.

3 Hypothesis Testing

In the remaining of this paper, in order to simplify the presentation, we assume that all Fitting programs are instantiated programs. Moreover, we use the term "program" to mean "Fitting program", unless explicitly stated otherwise.

3.1 Interpretations

First, we introduce some terminology and notation that we shall use throughout the paper. Given a program $\mathcal{P}$, call *interpretation* of $\mathcal{P}$ any function I over the Herbrand base $\mathcal{HB}_\mathcal{P}$ such that, for every atom A of $\mathcal{HB}_\mathcal{P}$, $I(A)$ is a value from $\mathcal{FOUR}$.

Two interpretations I and J are *compatible* if, for every ground atom A, $(I(A) \neq \mathcal{U}$ and $J(A) \neq \mathcal{U}) \Rightarrow I(A) = J(A)$.

An interpretation I is a *part of* an interpretation J, denoted $I \leq J$, if $I(A) \neq \mathcal{U}$ implies $I(A) = J(A)$, for every ground atom A. Clearly, the part-of relation just defined is a partial ordering on the set $\mathcal{V}(\mathcal{FOUR})$ of all interpretations over $\mathcal{FOUR}$. Given an interpretation I, we denote by $def(I)$ the set of all ground atoms A such that $I(A) \neq \mathcal{U}$. Moreover, if S is any set of ground atoms, we define the *restriction* of I to S, denoted by $I_{/S}$ as follows: for all $A \in \mathcal{HB}_{\mathcal{P}}$,

$$I_{/S}(A) = \begin{cases} I(A) & \text{if } A \in S, \\ \mathcal{U}, & \text{otherwise.} \end{cases}$$

The operations of $\mathcal{FOUR}$ can be extended naturaly to $\mathcal{V}(\mathcal{FOUR})$ in the following way: $I \oplus J(A) = I(A) \oplus J(A)$ and similarly for the other operations.

The actions of interpretations can be extended from atoms to formulas as follows:

- $I(X \wedge Y) = I(X) \wedge I(Y)$, and similarly for the other operators,
- $I((\exists x)\phi(x)) = \bigvee_{t=closedterm} I(\phi(t))$, and
- $I((\forall x)\phi(x)) = \bigwedge_{t=closedterm} I(\phi(t))$.

If B is a closed formula then we say that B evaluates to the logical value α, with respect to an interpretation I, denoted by $B \equiv \alpha$ w.r.t. I or by $B \equiv_I \alpha$, if $J(B) = \alpha$ for any interpretation J such that $I \leq J$ (i.e. if the value of B is equal to α with respect to the defined atoms of I whatever the values of underdefined atoms could be). There are formulas B in which underdefined atoms do not matter for the logical value that can be associated with B. For example let us take $B = A \vee C$ and let the interpretation I be defined by $I(A) = \mathcal{U},\ I(C) = \mathcal{T}$; then no matter how A is interpreted B is evaluated to $\mathcal{T}$, that is, $B \equiv_I \mathcal{T}$.

Given an interpretation I let $I_{\mathcal{O}}$ be the interpretation defined by : if $I(A) \neq \mathcal{U}$ then $I_{\mathcal{O}}(A) = I(A)$ else $I_{\mathcal{O}}(A) = \mathcal{O}$, for every atom A. The following lemma provides a method of testing whether $B \equiv_I \alpha$, based on the interpretation $I_{\mathcal{O}}$.

Lemma 1. *Given a closed formula B, $B \equiv_I \alpha$ iff $I(B) = \alpha$ and $I_{\mathcal{O}}(B) = \alpha$.*

3.2 The Support of a Hypothesis

Given a program $P = \langle F, R \rangle$ we consider two ways of inferring information from $\mathcal{P}$. First by activating the rules of R in order to derive new facts from those of F, through an immediate consequence operator T. Second, by a kind of default reasoning based on a given hypothesis.

The immediate consequence operator T that we use takes as input the facts of F and returns an interpretation $T(F)$, defined as follows: for all ground atoms A,

$$T_R(F)(A) = \begin{cases} \alpha \text{ if } A \leftarrow B \in R \text{ and } B \equiv_F \alpha \\ \mathcal{U}, \text{ otherwise} \end{cases}$$

What we call a *hypothesis* is actually just an interpretation H. However, we use the term "hypothesis" to stress the fact that the values assigned by H to the atoms of the Herbrand base are *assumed* values - and *not* values that have

been computed using the facts and rules of the program. As such, a hypothesis H must be tested against the "sure" knowledge provided by $\mathcal{P}$. The test consists of "adding" H to F, then activating the rules of $\mathcal{P}$ (using T) to derive an interpretation H'. If $H \leq H'$, then the hypothesis H is a sound one, i.e. the values defined by H are not in contradiction with those defined by $\mathcal{P}$. Hence the following definition:

Definition 2 (Sound Hypothesis). *Let $\mathcal{P} = \langle F, R\rangle$ be a program and H a hypothesis. H is sound w.r.t. $\mathcal{P}$ if*

- *F and H are compatible, and*
- *$H_{/Head(\mathcal{P})} \leq T(F \oplus H)$, where $Head(\mathcal{P}) = \{\mathcal{A} \mid \exists \mathcal{A} \leftarrow \mathcal{B} \in \mathcal{P}\}$.*

We use the restriction of H to $Head(\mathcal{P})$ before making the comparison with $T(F \oplus H)$ because all atoms which are not head of any rule of $\mathcal{P}$ will be assigned to the value $Underdefined$ by $T(F \oplus H)$. Then H and $T(F \oplus H)$ are compatible on these atoms.

Even if a hypothesis H is not sound w.r.t. $\mathcal{P}$, it may be that some part of H is sound w.r.t. $\mathcal{P}$. Of course, we are interested to know what is the maximal part of H that is sound w.r.t. $\mathcal{P}$. We shall call this maximal part the "support" of H. To see that the maximal part of H is unique (and thus that the support is a well-defined concept), we give the following lemma:

Lemma 2. *If H_1 and H_2 are two sound parts of H w.r.t. $\mathcal{P}$, then $H_1 \oplus H_2$ is sound w.r.t. $\mathcal{P}$.*

Thus the maximal sound part of H is defined by $\bigoplus\{H' \mid H' \leq H$ and H' is sound w.r.t. $\mathcal{P}\}$.

Definition 3 (Support). *Let $\mathcal{P}$ be a program and H a hypothesis. The support of H w.r.t. $\mathcal{P}$, denoted $s^H_{\mathcal{P}}$, is the maximal sound part of H w.r.t. $\mathcal{P}$ (where maximality is understood w.r.t. the part-of ordering $\leq$).*

We now give an algorithm for computing the support $s^H_{\mathcal{P}}$ of a hypothesis H w.r.t. a program $\mathcal{P}$.

Consider the following sequence $\langle PF_i\rangle,\ i \geq 0$:

- $PF_0 = \emptyset$;
- $PF_{i+1} = \{A \mid A \leftarrow B \in \mathcal{P}$ and $B \not\equiv H(A)$ *w.r.t.* $F \oplus H_{/(\mathcal{HB}_{\mathcal{P}} \setminus PF_i) \setminus def(F)}\}$ for all $i \geq 0$,

The intuition here is that we want to evaluate step by step the atoms that could potentially have a logical value different than their values in H. We have the following results:

Proposition 1. *The sequence $\langle PF_i\rangle,\ i \geq 0$ is increasing with respect to set inclusion and it has a limit reached in a finite number of steps. This limit is denoted PF.*

Theorem 1. *Let $\mathcal{P}$ and H be fixed. Then $s^H_{\mathcal{P}} = H_{/(\mathcal{HB}_{\mathcal{P}} \setminus PF)}$.*

4 Hypothesis Based Semantics

As we explained earlier, given a program $P = \langle F, R \rangle$, we derive information in two ways: by activating the rules (i.e. by applying the immediate consequence operator T) and by making a hypothesis H and computing its support $s^H_{\mathcal{P}}$ w.r.t. $\mathcal{P}$. In the whole, the information that we derive comes from $T(F) \oplus s^H_{\mathcal{P}}$.

Proposition 2. *The sequence* $\langle F_n \rangle$, $n \geq 0$ *defined by* $F_0 = F$ *and* $F_{n+1} = T_R(F_n) \oplus s^H_{\langle F_n, R \rangle}$ *is increasing with respect to* $\leq$, *so it has a limit denoted by* $sem^H_{\mathcal{P}}$.

We recall that an interpretation I is a model of a program $\mathcal{P}$ if for every rule $A \leftarrow B$ of $\mathcal{P}$, $I(B) \leq_t I(A)$.

Proposition 3. *The interpretation* $sem^H_{\mathcal{P}}$ *is a model of* $\mathcal{P}$.

This justifies the following definition of semantics for $\mathcal{P}$.

Definition 4 (H-semantics of $\mathcal{P}$). *The interpretation* $sem^H_{\mathcal{P}}$ *is defined to be the semantics of* $\mathcal{P}$ *w.r.t.* H *or the* H*-semantics of* $\mathcal{P}$.

Following this definition, any given program $\mathcal{P}$ can be associated with different semantics, one for each possible hypothesis H. Theorem 2 below asserts that this approach extends the usual semantics of Datalog programs with negation to a broader class of programs, namely the Fitting programs.

Two remarks are in order here before stating Theorem 2. First, if we restrict our attention to three values only, i.e. $\mathcal{F}$, $\mathcal{T}$ and $\mathcal{U}$, then our definition of interpretation is equivalent to the one used by Van Gelder et als [11], in the following sense: given an interpretation I following our definition, the set $\{A \mid I(A) = \mathcal{T}\} \cup \{\neg A \mid I(A) = \mathcal{F}\}$ is a partial interpretation following [11]; conversely, given a partial interpretation J following [11], the function I defined by: $I(A) = \mathcal{T}$ if $A \in J$, $I(A) = \mathcal{F}$ if $\neg A \in J$, and $I(A) = \mathcal{U}$ otherwise, is an interpretation in our sense.

Second, if we restrict our attention to Datalog programs with negation (recall that the class of Fitting programs strictly contains the Datalog programs with negation) then the concept of sound interpretation for the everywhere false hypothesis reduces to that of unfounded set of Van Gelder et als [11]. The difference is that the definition in [11] has rather a syntactic flavor, while ours has a semantic flavor. Moreover, our definition not only extends the concept of Unfounded set to four-valued logic, but also generalizes its definition to any given hypothesis H (not just the everywhere false hypothesis).

Theorem 2. *Let* $\mathcal{P}$ *be a Datalog programs with negation.*

1. *If* $H_{\mathcal{F}}$ *is the everywhere false hypothesis, then* $sem^{H_{\mathcal{F}}}_{\mathcal{P}}$ *coincides with the well-founded semantics of* $\mathcal{P}$*;*
2. *If* $H_{\mathcal{U}}$ *is the everywhere underdefined hypothesis, then* $sem^{H_{\mathcal{U}}}_{\mathcal{P}}$ *coincides with the Kripke-Kleene semantics of* $\mathcal{P}$.

5 Concluding Remarks

We have defined a formal framework for information integration based on hypothesis testing. A basic concept of this framework is the support provided by a program $\mathcal{P} = \langle F, R \rangle$ to a hypothesis H. The support of H is the maximal part of H that does not contradict the facts of F or the facts derived from F using the rules of R.

We have then used the concept of support to define hypothesis-based semantics for the class of Fitting programs, and we have given an algorithm for computing these semantics.

Finally, we have shown that our semantics extends the well-founded semantics and the Kripke-Kleene semantics to Belnap's four-valued logic, and also generalizes them in the following sense: if we restrict our attention to three-valued logics then for $H_{\mathcal{F}}$ the everywhere $false$ interpretation our semantics reduces to the well-founded semantics, and for $H_{\mathcal{U}}$ the everywhere $underdefined$ interpretation our semantics reduces to the Kripke-Kleene semantics.

We believe that hypothesis-based semantics can be useful not only in the context of information integration but also in the context of explanation-based systems. Indeed, assume that a given hypothesis H turns out to be a part of the H-semantics of a program $\mathcal{P}$. Then $\mathcal{P}$ can be seen as an "explanation" of the hypothesis H. We are currently investigating several aspects of this explanation oriented viewpoint.

References

1. *Data Warehousing, Flexible Views for Ever-changing Corporate Data*, Communications of the ACM, volume 41, number 9, septembre 1998.
2. Belnap, N. D., Jr, *A Useful Four-Valued Logic*, in: J. M. Dunn and G. Epstein (eds.), Modern Uses of Multiple-valued Logic, D. Reichel, Dordrecht, 1977.
3. Bidoit N., Froideveaux C., *Negation by default and unstratifiable logic programs*, TCS, 78, (1991)
4. Fitting, M. C., *A Kripke/Kleene Semantics for Logic Programs*, J. Logic Programming, 2:295-312 (1985).
5. Fitting, M. C., *Bilattices and the Semantics of Logic Programming*, J. Logic Programming, 11:91-116 (1991).
6. Fitting, M. C., *The Family of Stable Models*, J. Logic Programming, 17:197-225 (1993).
7. Fuhr, N. and Rölleke, T., *HySpirit – a Probabilistic Inference Engine for Hypermedia Retrieval in Large Databases*, (1997).
8. Ginsberg, M. L., *Multivalued Logics: a Uniform Approach to Reasoning in Artificial Intelligence*, Computational Intelligence, 4:265-316, 1988.
9. Ginsberg, M. L., *Bilattices and modal operators*, J. of Logic Computation, 1:41-69, 1990.
10. Loyer, Y., Spyratos, N., Stamate, D., *Computing and Comparing Semantics of Programs in Four-valued Logics*, in: Proceedings of the 24th Symposium on Mathematical Foundations of Computer Science ($MFCS'99$), LNCS 1672, Springer Verlag, 1999.

11. Van Gelder, A., Ross, K. A., Schlipf, J. S., *Unfounded Set and Well-Founded Semantics for General Logic Programs*, in: Proceedings of the Seventh Symposium on Principles of Database Systems, 221-230, 1988.
12. Van Gelder, A., Ross, K. A., Schlipf, J. S., *The Well-Founded Semantics for General Logic Programs*, J. ACM, 38:620-650, 1991.
13. Zaniolo, C., *Database Relations with Null Values*, Journal of Computer and System Sciences, 28: 142-166, 1984.

Masaccio*: A Formal Model for Embedded Components**

Thomas A. Henzinger

University of California, Berkeley
tah@eecs.berkeley.edu

Abstract. Masaccio is a formal model for hybrid dynamical systems which are built from atomic discrete components (difference equations) and atomic continuous components (differential equations) by parallel and serial composition, arbitrarily nested. Each system component consists of an interface, which determines the possible ways of using the component, and a set of executions, which define the possible behaviors of the component in real time.

We formally define a class of entities called "components." The intended use of components is to provide a formal, structured model for software and hardware that interacts with a physical environment in real time. The model is formal in that it defines a component as a mathematical object, which can be analyzed. The model is structured in that it permits the hierarchical definition of a component, and the hierarchy can be exploited for structuring the analysis. Components are built from atomic components using six operations: parallel composition, serial composition, renaming of variables (data), renaming of locations (control), hiding of variables, and hiding of locations. There are two kinds of atomic components. An atomic discrete component is a difference equation which governs the instantaneous change of state. An atomic continuous component is a differential equation, which governs the evolutionary change of state over time. The mathematical semantics of a component is given by its *interface* and its *set of executions*. The interface of a component determines how the component can interact (be composed) with other components. Each execution specifies a possible behavior of the component as a sequence of instantaneous and evolutionary state changes.

The interface of a component Data enters and exits a component through variables; control enters and exits through locations. All variables are assumed to be typed, with domains such as the booleans $\mathbb{B}$, the nonnegative integers $\mathbb{N}$, and the reals $\mathbb{R}$. For each variable x, we assume that there is a primed version x' which has the same type as x. For a set V of variables, we denote by $[V]$ the set of type-conforming value assignments to the variables in V: if $x \in V$ and $q \in [V]$, then $q(x)$ is the value assigned by q to x. The *interface of a component* A consists of five parts:

* Version 1.0 (May 2000).

** This research was supported in part by the DARPA grants NAG2-1214 and F33615-C-98-3614, and by the MARCO grant 98-DT-660.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 549–563, 2000.
© Springer-Verlag Berlin Heidelberg 2000

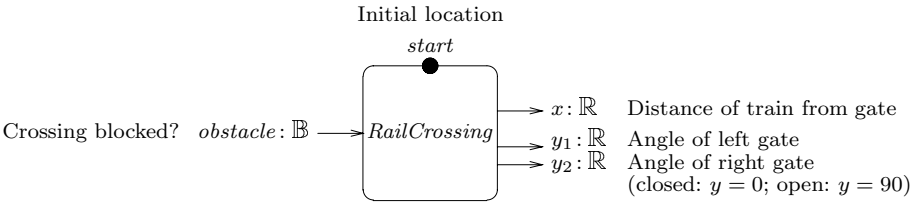


Fig. 1. A railroad crossing (safety: always $[|x| < 100 \Rightarrow y_1 = y_2 = 0]$)

- A finite set V_A^{in} of *input variables*. We write $V_A^{in'}$ for the set of primed variables whose unprimed versions are input variables.
- A finite set V_A^{out} of *output variables*. We require that the input and output variables are disjoint; that is, $V_A^{in} \cap V_A^{out} = \emptyset$. We refer to the collection $V_A^{in,out} = V_A^{in} \cup V_A^{out}$ of input and output variables as *I/O variables*. The value assignments in $[V_A^{in,out}]$ are called *I/O states*. Given an I/O state q, we denote by q' the value assignment in $[V_A^{in'}]$ which is derived from q in the following way: $q'(x') = q(x)$ for all input variables $x \in V_A^{in}$.
- A binary relation $\prec_A \subseteq V_A^{in,out} \times V_A^{out}$ of *dependencies* between I/O variables and output variables. The value of an output variable y can depend on previous values of any I/O variable x; intuitively, if $x \prec_A y$, then the value of y can depend, without delay, also on the concurrent value of x. A set U of I/O variables is *dependency-closed* if for all $x, y \in V_A^{in,out}$, if $x \prec_A y$ and $y \in U$, then $x \in U$. For example, the set V_A^{in} of input variables is dependency-closed.
- A finite set L_A^{intf} of *interface locations*. These are the locations through which control can enter or exit the component A.
- For each interface location $a \in L_A^{intf}$, a predicate $\varphi_A^{en}(a)$ on the variables in $V_A^{in,out} \cup V_A^{in'}$. Thus, given two I/O states p and q, the *entry condition* $\varphi_A^{en}(a)$ is either true or false at (p, q'), i.e., if each unprimed variable $x \in V_A^{in,out}$ is assigned the value $p(x)$, and each primed variable $y' \in V_A^{in'}$ is assigned the value $q'(y')$. Intuitively, if the current I/O state is p, and the input portion of the next I/O state is q', then the component A can be entered at location a iff the entry condition $\varphi_A^{en}(a)$ is true at (p, q').

We will distinguish between discrete and hybrid components. If A is a discrete component, then all I/O variables of A have discrete types, such as $\mathbb{B}$ or $\mathbb{N}$. Hybrid components have also I/O variables of type $\mathbb{R}$.

The executions of a component The possible finite behaviors of a component are called executions. Consider a component A. A *jump* of A is a pair $(p, q) \in [V_A^{in,out}] \times [V_A^{in,out}]$ of I/O states. The observation p is called the *source* of the jump, and q is the *sink*. A *flow* of A is a pair (δ, f) consisting of a positive real $\delta \in \mathbb{R}_{>0}$, and a function $f : \mathbb{R} \to [V_A^{in,out}]$ from the reals to I/O states which

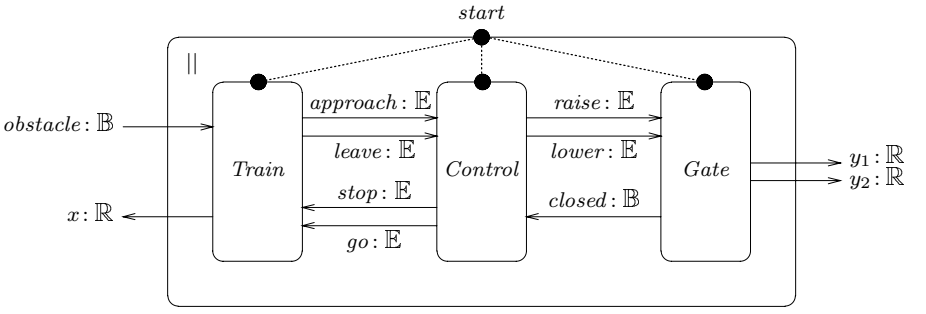


Fig. 2. The component *RailCrossing*

is differentiable[3] on the compact interval $[0, \delta] \subset \mathbb{R}$. The real δ is called the *duration* of the flow, the observation $f(0)$ is the *source*, and the observation $f(\delta)$ is the *sink*. A *step* of A is either a jump or a flow of A. The step w is *successive* to the step v if the sink of v is equal to the source of w. An *execution* of A is either a pair $(a, \boldsymbol{w})$ or a triple $(a, \boldsymbol{w}, b)$, where $a, b \in L_A^{intf}$ are interface locations and $\boldsymbol{w} = w_0 \cdots w_n$ is a nonempty, finite sequence of steps of A such that (1) the first step w_0 is a jump, and (2) each subsequent step w_i, for $1 \leq i \leq n$, is successive to the immediately preceding step w_{i-1}. The location a is called the *origin* of the execution, the sequence $\boldsymbol{w}$ is the *trace*, and the location b (when present) is the *destination*. If A is a discrete component, then all traces of A consist of jumps only; the traces of hybrid components contain also flows. We write E_A for the set of executions of the component A. We require that E_A is prefix-closed, deadlock-free, and input-permissive. Prefix closure ensures that the executions of a component can be generated operationally in a stepwise manner. The set E_A of executions is *prefix-closed* if the following four conditions are satisfied:

1. If $(a, \boldsymbol{w}, b) \in E_A$, then $(a, \boldsymbol{w}) \in E_A$.
2. If $(a, w_0 \cdots w_n) \in E_A$ for $n \geq 1$, then $(a, w_0 \cdots w_{n-1}) \in E_A$.
3. If $(a, \boldsymbol{w} \cdot (\delta, f)) \in E_A$ for a flow (δ, f), then $(a, \boldsymbol{w} \cdot (\varepsilon, f)) \in E_A$ for all reals $\varepsilon \in (0, \delta)$.
4. If $(a, (p, q)) \in E_A$ for a jump (p, q), then the entry condition $\varphi_A^{en}(a)$ is true at (p, q').

Deadlock freedom ensures that the stepwise generation of executions cannot deadlock inside a component. The set E_A of executions is *deadlock-free* if the following two conditions are satisfied:

1. For all interface locations a and I/O states p, if the entry condition $\varphi_A^{en}(a)$ is true at (p, q') for some I/O state q, then $(a, (p, q)) \in E_A$ for some jump (p, q). In other words, if the entry condition of location a is satisfiable at the I/O state p, then there is an execution with origin a and source p.

[3] On types other than $\mathbb{R}$, it can be assumed that only the constant functions are differentiable.

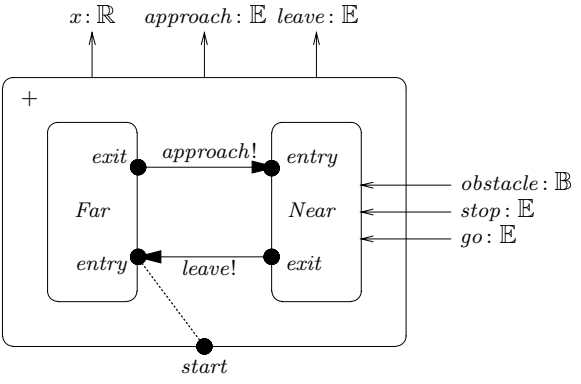


Fig. 3. The component *Train*

2. If $(a, \boldsymbol{w}) \in E_A$, then either $(a, \boldsymbol{w}, b) \in E_A$ for some interface location b, or $(a, \boldsymbol{w} \cdot (p, q)) \in E_A$ for some jump (p, q). In other words, every execution which does not have a destination can be prolonged by either a destination or a jump.

Input permissiveness ensures that a component cannot constrain the behavior of input variables. The set E_A of executions is *input-permissive* if the following two conditions are satisfied:

1. If $(a, (p, q_1)) \in E_A$ for a jump (p, q_1), then for every dependency-closed set U of I/O variables, and every I/O state q_2 such that (1) the I/O state q_2 agrees with q_1 on the variables in U and (2) the entry condition $\varphi_A^{en}(a)$ is true at (p, q_2'), there is an execution $(a, (p, q)) \in E_A$ whose sink q agrees with q_2 on the variables in U and the input variables.
2. If $(a, \boldsymbol{w} \cdot (p, q_1)) \in E_A$ for a nonempty trace $\boldsymbol{w}$ and a jump (p, q_1), then for every dependency-closed set U of I/O variables, and every I/O state q_2 which agrees with q_1 on the variables in U, there is an execution $(a, \boldsymbol{w} \cdot (p, q)) \in E_A$ whose sink q agrees with q_2 on the variables in U and the input variables.

If two components A and B have the same interface, then they can take each other's place in all contexts. We say that A *refines* (or *implements*) B if (1) the components A and B have the same interface and (2) every execution of A is also an execution of B; that is, $E_A \subseteq E_B$. If A refines B, then B can be thought of as a more abstract (permissive) version of A, with some details (constraints) left out in B which are spelt out in A. Since the executions of A are deadlock-free, if B has an execution with origin a, and A refines B, then A must also have an execution with origin a. Thus a component with a nonempty set of executions cannot be trivially implemented by a component with the empty set of executions.

The parallel composition of components Two components A and B *can be composed in parallel* if their interfaces satisfy the following three conditions:

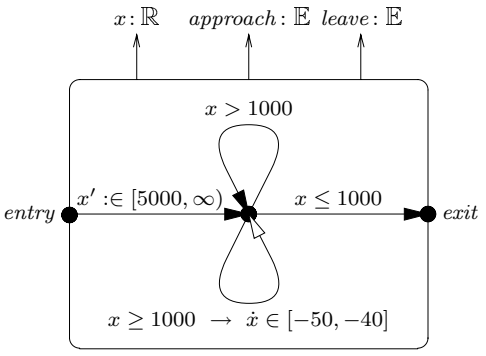


Fig. 4. The component *Far*

1. The output variables of A and B are disjoint; that is, $V_A^{out} \cap V_B^{out} = \emptyset$.
2. There is no inferred mutual dependency between an output variable of A and an output variable of B; that is, there do not exist two variables $x \in V_A^{out}$ and $y \in V_B^{out}$ such that both $x \prec_B^* y$ and $y \prec_A^* x$, where $\prec^*$ is the transitive closure of the dependency relation $\prec$.
3. For each interface location a common to both A and B, the entry conditions of a are equivalent in A and B; that is, if $a \in L_A^{intf} \cap L_B^{intf}$, then the entry condition $\varphi_A^{en}(a)$ is equivalent to the entry condition $\varphi_B^{en}(a)$. This implies, in particular, that $\varphi_A^{en}(a)$ does not constrain the primed outputs of B, nor does $\varphi_B^{en}(a)$ constrain the primed outputs of A.

If the components A and B can be composed in parallel, then $A||B$ is again a component. The *interface of the component* $A||B$ is defined from the interfaces of the subcomponents A and B:

- A variable is an input to $A||B$ if it is an input to A but not an output of B, or an input to B but not an output of A; that is, $V_{A||B}^{in} = (V_A^{in} \backslash V_B^{out}) \cup (V_B^{in} \backslash V_A^{out})$.
- A variable is an output of $A||B$ if it is an output of A or an output of B; that is, $V_{A||B}^{out} = V_A^{out} \cup V_B^{out}$.
- The dependencies of $A||B$ are inherited from both A and B; that is, $\prec_{A||B} = \prec_A \cup \prec_B$.
- The interface locations of $A||B$ are the interface locations of A together with the interface locations of B; that is, $L_{A||B}^{intf} = L_A^{intf} \cup L_B^{intf}$.
- If a is an interface location of both subcomponents A and B, then they agree on the entry condition, and this is also the entry condition of $A||B$; that is, if $a \in L_A^{intf} \cap L_B^{intf}$, then $\varphi_{A||B}^{en}(a) = (\exists V_A')\, \varphi_A^{en}(a) = (\exists V_B')\, \varphi_B^{en}(a)$, where $x' \in V_A'$ iff $x \in V_A^{in} \cap V_B^{out}$, and $y' \in V_B'$ iff $y \in V_B^{in} \cap V_A^{out}$. It follows that the component $A||B$ can be entered at location a iff both subcomponents A and B can be entered concurrently at a. The quantifiers (whose force, existential or universal, is immaterial) ensure syntactically that no primed

output variables occur freely in entry conditions. All other interface locations of $A||B$ have the unsatisfiable entry condition; that is, if $a \in L_A^{intf} \backslash L_B^{intf}$ or $a \in L_B^{intf} \backslash L_A^{intf}$, then $\varphi_{A||B}^{en}(a) = \mathit{false}$. These locations can be used only to exit the component $A||B$.

The *executions of the component* $A||B$ are defined from the executions of the subcomponents A and B:

- The pair $(a, \boldsymbol{w})$ is an execution of $A||B$ iff $(a, \boldsymbol{w}|_A)$ is an execution of A and $(a, \boldsymbol{w}|_B)$ is an execution of B, where $\boldsymbol{w}|_C$ is the restriction of the trace $\boldsymbol{w}$ to values for the I/O variables of the component C.
- The triple $(a, \boldsymbol{w}, b)$ is an execution of $A||B$ iff either $(a, \boldsymbol{w}|_A, b)$ is an execution of A and $(a, \boldsymbol{w}|_B)$ is an execution of B, or $(a, \boldsymbol{w}|_B, b)$ is an execution of B and $(a, \boldsymbol{w}|_A)$ is an execution of A.

In other words, parallel composition acts conjunctively on traces. In particular, each jump of A corresponds to a concurrent jump of B, and each flow of A corresponds to a concurrent flow of B with the same duration. If an execution of A reaches a destination, then the concurrent execution of B is terminated; if B reaches a destination, then the concurrent execution of A is terminated; if both A and B simultaneously reach destinations, then one of the two destinations is chosen nondeterministically. Note that the operator $||$ for parallel composition is associative and commutative. Furthermore, the refinement relation is preserved by parallel composition: if A and B are two components with the same interface, if A refines B, and if A (and therefore also B) can be composed in parallel with a component C, then $A||C$ refines $B||C$.

The serial composition of components Two components A and B *can be composed in series* if their interfaces agree on the output variables; that is, $V_A^{out} = V_B^{out}$. If the components A and B can be composed in series, then $A+B$ is again a component. The *interface of the component* $A+B$ is defined from the interfaces of the subcomponents A and B:

- A variable is an input to $A+B$ if it is an input to A or an input to B; that is, $V_{A+B}^{in} = V_A^{in} \cup V_B^{in}$.
- As A and B agree on their outputs, these are also the outputs of $A+B$; that is, $V_{A+B}^{out} = V_A^{out} = V_B^{out}$.
- The dependencies of $A+B$ are inherited from both A and B; that is, $\prec_{A+B} = \prec_A \cup \prec_B$.
- The interface locations of $A+B$ are the interface locations of A together with the interface locations of B; that is, $L_{A+B}^{intf} = L_A^{intf} \cup L_B^{intf}$.
- If a is an interface location of both A and B, then the entry condition of a in $A+B$ is the disjunction of the entry conditions of a in the subcomponents A and B; that is, if $a \in L_A^{intf} \cap L_B^{intf}$, then $\varphi_{A+B}^{en}(a) = \varphi_A^{en}(a) \vee \varphi_B^{en}(a)$. If a is an interface location of A but not of B, then the entry condition of a in $A+B$ is inherited from A; that is, if $a \in L_A^{intf} \backslash L_B^{intf}$, then $\varphi_{A+B}^{en}(a) = \varphi_A^{en}(a)$. If a is an interface location of B but not of A, then the entry condition of a in

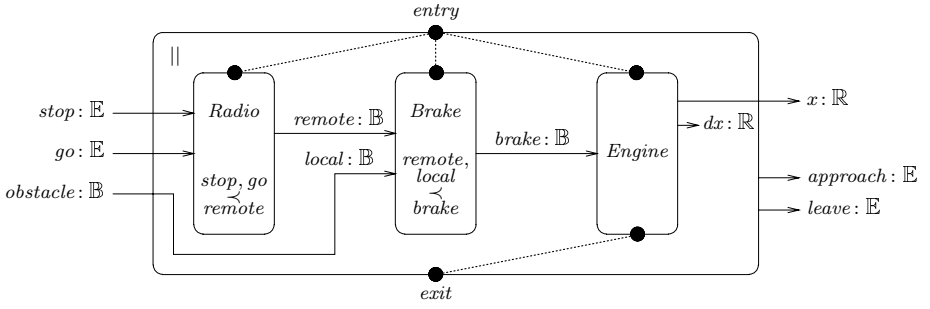


Fig. 5. The component *Near*

$A+B$ is inherited from B; that is, if $a \in L_B^{intf} \backslash L_A^{intf}$, then $\varphi_{A+B}^{en}(a) = \varphi_B^{en}(a)$. This is because the component $A + B$ is entered at location a iff either subcomponent A or subcomponent B is entered at a.

The *executions of the component* $A + B$ are defined from the executions of the subcomponents A and B:

- The pair $(a, \boldsymbol{w})$ is an execution of $A + B$ iff either $(a, \boldsymbol{w}|_A)$ is an execution of A, or $(a, \boldsymbol{w}|_B)$ is an execution of B.
- The triple $(a, \boldsymbol{w}, b)$ is an execution of $A + B$ iff either $(a, \boldsymbol{w}|_A, b)$ is an execution of A, or $(a, \boldsymbol{w}|_B, b)$ is an execution of B.

In other words, serial composition acts disjunctively on traces. Note that the operator $+$ for serial composition is associative, commutative, and idempotent. Furthermore, the refinement relation is preserved by serial composition: if A and B are two components with the same interface, if A refines B, and if A (and therefore also B) can be composed in series with a component C, then $A + C$ refines $B + C$.

Variable renaming When constructing a parallel composition $A||B$, inputs of A can be identified with outputs of B, and vice versa, by renaming variables. The variable x *can be renamed to* y in component A if x is an I/O variable of A and y is different from all I/O variables of A; that is, $x \in V_A^{in,out}$ and $y \notin V_A^{in,out}$. If x can be renamed to y in A, then $A[x := y]$ is again a component. The *interface of the component* $A[x := y]$ is defined from the interface of A: let $V_{A[x:=y]}^{in} = (V_A^{in} \backslash \{x\}) \cup \{y\}$, let $V_{A[x:=y]}^{out} = (V_A^{out} \backslash \{x\}) \cup \{y\}$, let $L_{A[x:=y]}^{intf} = L_A^{intf}$, and let $\prec_{A[x:=y]}$ and $\varphi_{A[x:=y]}^{en}$ result from renaming x to y in $\prec_A$ and in φ_A^{en}, respectively. The *executions of the component* $A[x := y]$ result from renaming x to y in the traces of the executions of A. The refinement relation is preserved by the renaming of variables: if A and B are two components with the same interface, if A refines B, and if x can be renamed to y in A (and therefore also in B), then $A[x := y]$ refines $B[x := y]$.

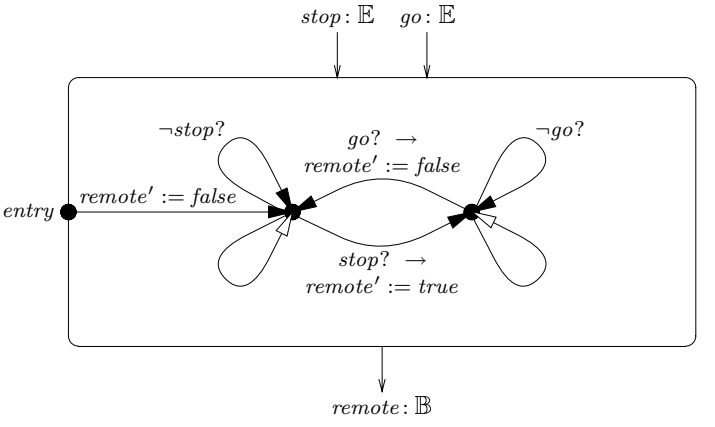


Fig. 6. The component *Radio*

Location renaming When constructing a serial composition $A + B$, interface locations of A can be identified with interface locations of B by renaming locations. The location a *can be renamed to* b in component A if a is an interface location of A; that is, $a \in L_A^{intf}$. The location b may or may not be an interface location of A. If a can be renamed to b in A, then $A[a := b]$ is again a component. The *interface of the component* $A[a := b]$ is defined from the interface of A: let $V_{A[a:=b]}^{in} = V_A^{in}$, let $V_{A[a:=b]}^{out} = V_A^{out}$, let $\prec_{A[a:=b]} = \prec_A$, let $L_{A[a:=b]}^{intf} = (L_A^{intf} \backslash \{a\}) \cup \{b\}$, let $\varphi_{A[a:=b]}^{en}(b) = \varphi_A^{en}(a)$ if $b \notin L_A^{intf}$, let $\varphi_{A[a:=b]}^{en}(b) = \varphi_A^{en}(a) \vee \varphi_A^{en}(b)$ if $b \in L_A^{intf}$, and let $\varphi_{A[a:=b]}^{en}(c) = \varphi_A^{en}(c)$ for all locations $c \in L_A^{intf} \backslash \{a, b\}$. Consequently, if both a and b are interface locations of A, then the component $A[a := b]$ can be entered at location b whenever the original component A can be entered at either a or b. The *executions of the component* $A[a := b]$ result from renaming a to b in the origins and destinations of the executions of A. The refinement relation is preserved by the renaming of locations: if A and B are two components with the same interface, if A refines B, and if a can be renamed to b in A (and therefore also in B), then $A[a := b]$ refines $B[a := b]$.

Variable hiding Hiding renders a variable local to a component, and invisible to the outside. Hidden variables do not maintain their values from one exit of a component to a subsequent entry, but they are nondeterministically reinitialized upon every entry to the component as to satisfy the applicable entry condition. The variable x *can be hidden* in the component A if x is an output variable of A; that is, $x \in V_A^{out}$. If x can be hidden in A, then $A \backslash x$ is again a component. The *interface of the component* $A \backslash x$ is defined from the interface of A: let $V_{A\backslash x}^{in} = V_A^{in}$, let $V_{A\backslash x}^{out} = V_A^{out} \backslash \{x\}$, let $\prec_{A\backslash x}$ be the intersection of the transitive closure $\prec_A^*$ with $V_{A\backslash x}^{in,out} \times V_{A\backslash x}^{out}$, let $L_{A\backslash x}^{intf} = L_A^{intf}$, and let $\varphi_{A\backslash x}^{en}(a) = (\exists x)\, \varphi_A^{en}(a)$

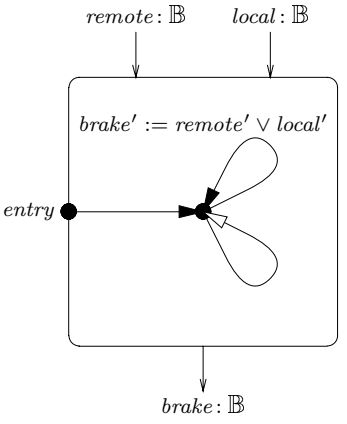


Fig. 7. The component *Brake*

for all locations $a \in L_A^{intf}$. In other words, upon entry of the component $A\backslash x$ at location a, the output variable x has an unknown value which permits the satisfaction of the entry condition $\varphi_A^{en}(a)$. The *executions of the component* $A\backslash x$ result from restricting the traces of the executions of A to values for the I/O variables of $A\backslash x$. Note that the component $A\backslash x\backslash y$ is identical to the component $A\backslash y\backslash x$. Furthermore, the refinement relation is preserved by the hiding of variables: if A and B are two components with the same interface, if A refines B, and if x can be hidden in A (and therefore also in B), then $A\backslash x$ refines $B\backslash x$.

Location hiding Hiding renders a location internal to a component, and inaccessible from the outside. The location c *can be hidden* in the component A if c is an interface location of A and the entry condition $\varphi_A^{en}(c)$ is valid; that is, $c \in L_A^{intf}$, and $\varphi_A^{en}(c)$ is equivalent to *true*. Consequently, an interface location c of A can be hidden only if the component A cannot deadlock at c, no matter what the current I/O state and the next inputs. If c can be hidden in A, then $A\backslash c$ is again a component. The *interface of the component* $A\backslash c$ is defined from the interface of A: let $V_{A\backslash c}^{in} = V_A^{in}$, let $V_{A\backslash c}^{out} = V_A^{out}$, let $\prec_{A\backslash c} = \prec_A$, let $L_{A\backslash c}^{intf} = L_A^{intf}\backslash\{c\}$, and let $\varphi_{A\backslash c}^{en}(a) = \varphi_A^{en}(a)$ for all locations $a \in L_{A\backslash c}^{intf}$. The *executions of the component* $A\backslash c$ are defined from the executions of A:

- The pair $(a, \boldsymbol{w})$ is an execution of $A\backslash c$ iff $c \neq a$ and either $(a, \boldsymbol{w})$ is an execution of A, or there is a finite sequence $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$ of traces, $n \geq 2$, such that $\boldsymbol{w} = \boldsymbol{w}_1 \cdots \boldsymbol{w}_n$ and the following are all executions of A: the triple $(a, \boldsymbol{w}_1, c)$, the triples $(c, \boldsymbol{w}_i, c)$ for all $1 < i < n$, and the pair $(c, \boldsymbol{w}_n)$.
- The triple $(a, \boldsymbol{w}, b)$ is an execution of $A\backslash c$ iff $c \notin \{a, b\}$ and either $(a, \boldsymbol{w}, b)$ is an execution of A, or there is a finite sequence $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n$ of traces, $n \geq 2$, such that $\boldsymbol{w} = \boldsymbol{w}_1 \cdots \boldsymbol{w}_n$ and the following are all executions of A: the triple $(a, \boldsymbol{w}_1, c)$, the triples $(c, \boldsymbol{w}_i, c)$ for all $1 < i < n$, and the triple $(c, \boldsymbol{w}_n, b)$.

In other words, the executions of $A\backslash c$ result from stringing together, at location c,

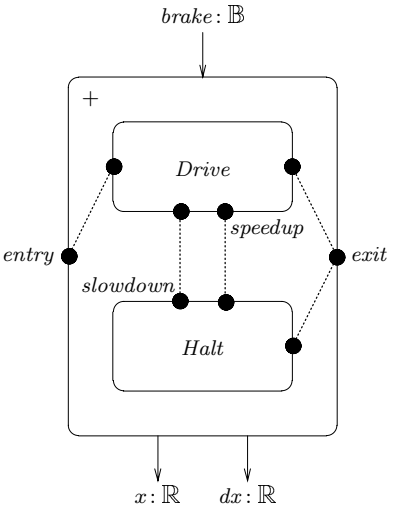


Fig. 8. The component *Engine*

a finite number of executions of A. Note that the component $A\backslash c\backslash d$ is identical to the component $A\backslash d\backslash c$. Furthermore, the refinement relation is preserved by the hiding of locations: if A and B are two components with the same interface, if A refines B, and if c can be hidden in A (and therefore also in B), then $A\backslash c$ refines $B\backslash c$.

Atomic discrete components The *discrete components* are built from atomic discrete components using the six operations of parallel and serial composition, variable and location renaming, and variable and location hiding. Each *atomic discrete component* is specified by a jump action. A *jump action* J consists of a finite set X_J of *source variables*, a finite set Y_J of *uncontrolled sink variables*, a finite set Z_J of *controlled sink variables* disjoint from Y_J, and a predicate φ_J^{jump} on the variables in $X_J \cup Y_J' \cup Z_J'$, where V' is the set of primed versions of the variables in V. The predicate φ_J^{jump} is called *jump predicate*; it is typically written as a guarded difference equation. The jump action J specifies the component $A(J)$. The *interface of the component* $A(J)$ is defined as follows:

- The inputs to $A(J)$ are the source variables of J which are not controlled sink variables, together with the uncontrolled sink variables; that is, $V_{A(J)}^{in} = (X_J \backslash Z_J) \cup Y_J$.
- The outputs of $A(J)$ are the controlled sink variables of J; that is, $V_{A(J)}^{out} = Z_J$.
- Each controlled sink variable depends on each uncontrolled sink variable; that is, for all $x \in V_{A(J)}^{in,out}$ and $y \in V_{A(J)}^{out}$, define $x \prec_{A(J)} y$ iff $x \in Y_J$ and $y \in Z_J$.
- The component $A(J)$ has two interface locations, say, *from* and *to*; that is,

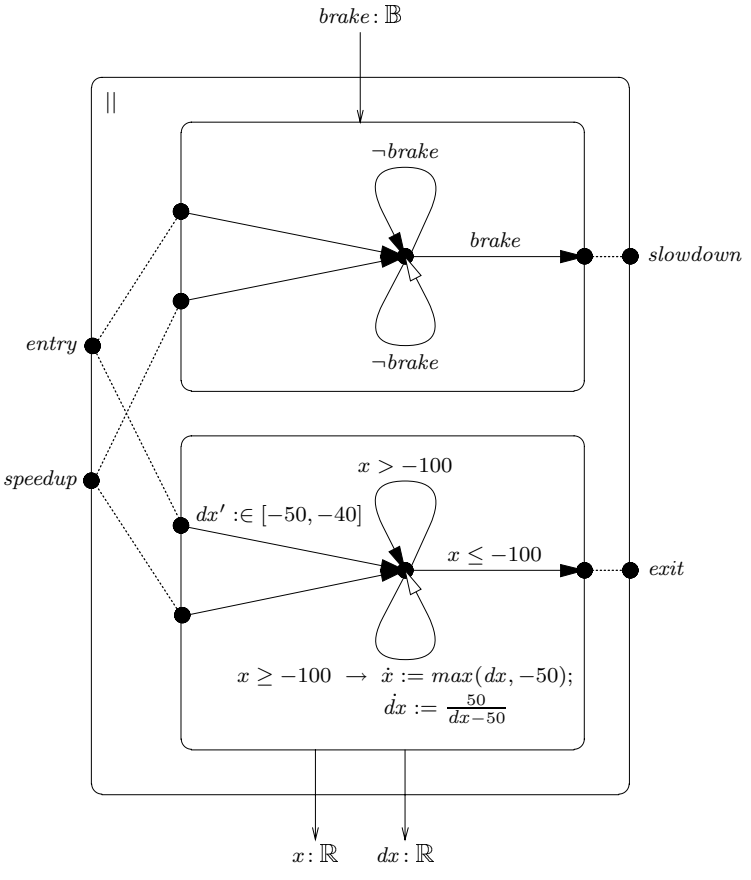


Fig. 9. The component *Drive*

$L^{intf}_{A(J)} = \{from, to\}$.

– The entry condition of *from* is the projection of the jump predicate to the source variables and the primed versions of the uncontrolled sink variables; that is, $\varphi^{en}_{A(J)}(from) = (\exists Z'_J)\, \varphi^{jump}_J$. The entry condition of *to* is unsatisfiable; that is, $\varphi^{en}_{A(J)}(to) = false$.

The *executions of the component* $A(J)$ are defined as follows: the pair $(a, \boldsymbol{w})$ is an execution of $A(J)$ iff $a = from$ and the trace $\boldsymbol{w}$ consists of a single jump (p, q) such that the jump predicate φ^{jump}_J is true if each source variable $x \in X_J$ is assigned the value $p(x)$, and each primed sink variable $y' \in Y'_J \cup Z'_J$ is assigned the value $q(y)$. Moreover, the triple $(a, \boldsymbol{w}, b)$ is an execution of $A(J)$ iff the pair $(a, \boldsymbol{w})$ is an execution of $A(J)$, and $b = to$. In other words, the traces of the atomic discrete component $A(J)$ are the jumps that satisfy the jump predicate φ^{jump}_J. From any given source, there may be no such jumps or there may be several.

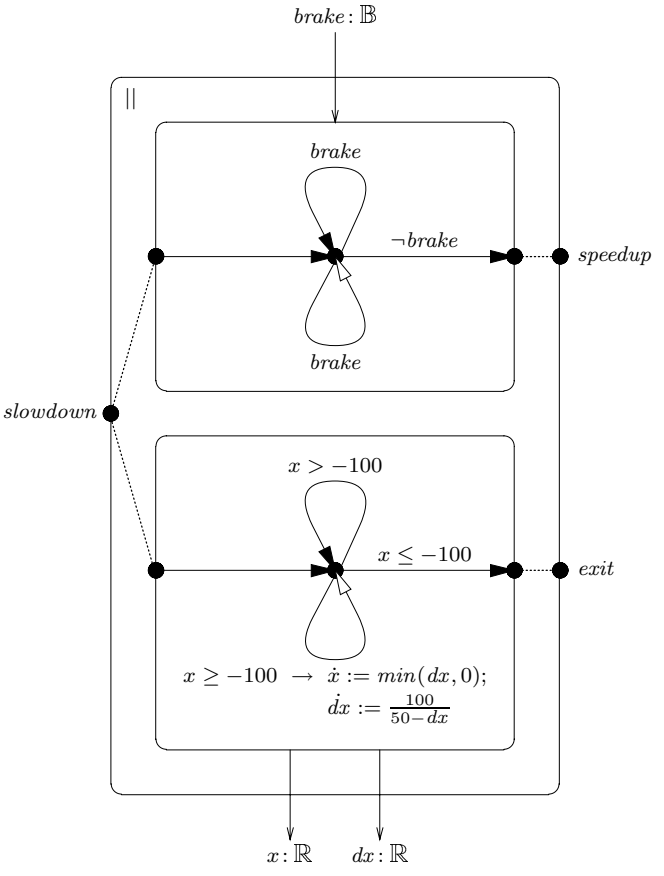


Fig. 10. The component *Halt*

Atomic continuous components The *hybrid components* are built from both atomic discrete components and atomic continuous components using the six operations on components. Each *atomic continuous component* is specified by a flow action. A *flow action* F consists of a finite set X_F of *source variables*, a finite set Y_F of *uncontrolled flow variables* of type $\mathbb{R}$, a finite set Z_F of *controlled flow variables* of type $\mathbb{R}$ disjoint from Z_F, and a predicate φ_F^{flow} on the variables in $X_F \cup \dot{Y}_F \cup \dot{Z}_F$, where $\dot{V}$ is the set of dotted versions of the variables in V. We use the notation $\dot{V}$ only if all variables in V have type $\mathbb{R}$, with the intent that the dotted variable $\dot{x} \in \dot{V}$ represents the first derivative of $x \in V$. The predicate φ_F^{flow} is called *flow predicate*; it is typically written as a guarded differential equation. The flow action F specifies the component $A(F)$. The *interface of the component* $A(F)$ is defined as follows:

– The inputs to $A(F)$ are the source variables of F which are not controlled flow variables, together with the uncontrolled flow variables; that is, $V_{A(F)}^{in} =$

$(X_F \backslash Z_F) \cup Y_F$.
- The outputs of $A(F)$ are the controlled flow variables of F; that is, $V^{out}_{A(F)} = Z_F$.
- Each controlled flow variable depends on each uncontrolled flow variable; that is, for all $x \in V^{in,out}_{A(F)}$ and $y \in V^{out}_{A(F)}$, define $x \prec_{A(F)} y$ iff $x \in Y_F$ and $y \in Z_F$.
- The component $A(F)$ has two interface locations, say, *from* and *to*; that is, $L^{intf}_{A(F)} = \{from, to\}$.
- The entry conditions of *from* and *to* are unsatisfiable; that is, $\varphi^{en}_{A(F)}(from) = \varphi^{en}_{A(F)}(to) = false$. This ensures that jumps take precedence over flows, in the sense that if a component A wishes to jump and concurrently another component B wishes to flow, then the parallel composition $A||B$ will jump.

The *executions of the component* $A(F)$ are defined as follows: the pair $(a, \boldsymbol{w})$ is an execution of $A(F)$ iff $a = from$ and the trace w consists of a single flow (δ, f) such that for all reals $\varepsilon \in [0, \delta]$, the flow predicate φ^{flow}_F is true if each source variable $x \in X_F$ is assigned the value $f(\varepsilon)(x)$, and each dotted flow variable $\dot{y} \in \dot{Y}_F \cup \dot{Z}_F$ is assigned the value $f'(\varepsilon)(y)$, where f' is the first derivative of f. Moreover, the triple $(a, \boldsymbol{w}, b)$ is an execution of $A(F)$ iff the pair $(a, \boldsymbol{w})$ is an execution of $A(F)$, and $b = to$. In other words, the traces of the atomic continuous component $A(F)$ are the flows that at all times satisfy the flow predicate φ^{flow}_F. From any given source and duration, there may be no such flows or there may be several. If there is a flow of a given duration, then there is a flow for each shorter duration as well.

Example The Figures 1–10 illustrate parts of a component which models the control of a railway crossing. In the figures we use the following conventions. Components are represented by rectangles. Input and output variables are represented, repectively, by arrows to and from component boundaries. Locations are represented by little black disks, and between locations, jump actions are represented by arrows with solid (black) points, and flow actions are represented by arrows with hollow (white) points. Interface locations are drawn on component boundaries. Variables which are identified by renaming are connected by solid lines; locations which are identified by renaming are connected by dotted lines. The event type $\mathbb{E}$ is similar to the boolean type $\mathbb{B}$, except that if a variable x has type $\mathbb{E}$, then it is of interest when the value of x changes (from *true* to *false*, or vice versa) whereas the actual value of x at any time is irrelevant. If x has type $\mathbb{E}$, then we write $x!$ for $x' := \neg x$ (to issue an event x), and $x?$ for $x' \neq x$ (to query the presence of an event x). Instead of using jump and flow predicates, we annotate jump and flow actions with guarded commands, because they allow us to omit specifying that a variable is left unchanged. Specifically, by default, an omitted guard is *true*, an omitted list of assignments is empty, the default jump assignment is $x' := x$, and the default flow assignment is $\dot{x} := 0$.

The component *RailCrossing* has three real outputs, the distance x of the train from the crossing, and the positions y_1 and y_2 of the two gates. The boolean input *obstacle* indicates whether or not the driver of the train sees an obstacle

on the crossing, in which case she will try to stop the train. The component *RailCrossing* is the parallel composition of three subcomponents, the train *Train*, the gate mechanics *Gate*, and the gate controller *Control*. We will look only into the component *Train*, which communicates with the gate controller via the output events *approach* and *leave*, and the input events *stop* and *go* (for example, if the gate fails, the gate controller may signal the train to stop). The component *Train* is the serial composition of four subcomponents: the component *Far* controls the speed $\dot{x}$ of the train when it is more than 1000 meters from the gate; an unnamed component issues the event *approach* when the train is at 1000 meters from the gate; the component *Near* controls the speed $\dot{x}$ of the train when it is between 1000 and -100 meters from the gate; and an unnamed component issues the event *leave* when the train is at -100 meters from the gate. The component *Far* holds the speed of the train between 40 and 50 meters per second. The component *Near* is the parallel composition of three subcomponents, *Radio*, *Brake*, and *Engine*. The component *Radio* translates *stop* and *go* events received from the gate controller into a boolean output *remote*, which causes the train to brake. The component *Brake* is an OR gate which computes the boolean disjunction *brake* of the two brake signals *remote* and *local*, where the latter is issued by the driver when she sees an obstacle. The component *Engine* controls the acceleration $\dot{dx} = \ddot{x}$ of the train. It does so by switching between the component *Drive*, which accelerates the train to 50 meters per second, and the component *Halt*, which causes the train to stop. The switching between *Drive* and *Halt* is controlled by the boolean input *brake*, and occurs through the locations *slowdown* and *speedup*. No matter whether the train is accelerating or braking, as soon as it is 100 meters past the gate, the component *Near* relinquishes control.

Acknowledgments Concurrent and sequential hierarchies have long been nested in informal and semiformal ways (e.g., Statecharts [Har87], UML [BRJ99]). While these languages enjoy considerable acceptance as good engineering practice, they do not support compositional formal analysis. The author was pointed to the importance of heterogeneous hierarchies, and the lack of adequate formalization, by Edward Lee and the Ptolemy project at UC Berkeley [DGH+99]. The proposed solution, Masaccio, is built by combining what the author believes are the key ingredients for achieving concurrent, sequential, and timed compositionality: much of the way Masaccio handles parallel composition is borrowed from Reactive Modules [AH99], which in turn build on other synchronous languages such as Esterel [BG88] and Signal [BlGJ91]; many ideas for structuring serial composition are inspired by the work of Rajeev Alur and Radu Grosu [AG00]; the formalization of hybrid executions using the dichotomy between jumps and flows is due to the research around Hybrid Automata [MMP92, ACH+95, AH97b]. Mixed discrete-continuous dynamics have been previously combined with both synchronous [AH97a] and semisynchronous [LSVW96] concurrency models; these settings, however, do not support the nesting of concurrency and sequencing. Related hybrid languages, which focus on simulation rather than mathematical semantics, include Shift [DGV96] and Charon

[AGH+00]. The author is grateful to the Fresco (Formal REal-time Software COmponents[4]) group at UC Berkeley, namely, Luca de Alfaro, Ben Horowitz, Ranjit Jhala, Rupak Majumdar, Freddy Mang, Christoph Meyer, Marius Minea, and Vinayak Prabhu, for many challenging and productive discussions.

References

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AG00] R. Alur and R. Grosu. Modular refinement of hierarchic reactive machines. In *Proceedings of the 27th Annual Symposium on Principles of Programming Languages*, pages 390–402. ACM Press, 2000.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in Charon. In *HSCC 00: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 1790. Springer-Verlag, 2000.

[AH97a] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR 97: Concurrency Theory*, Lecture Notes in Computer Science 1243, pages 74–88. Springer-Verlag, 1997.

[AH97b] R. Alur and T.A. Henzinger. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer*, 1:86–109, 1997.

[AH99] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[BG88] G. Berry and G. Gonthier. The synchronous programming language Esterel: design, semantics, implementation. Technical Report 842, INRIA, 1988.

[BlGJ91] A. Benveniste, P. le Guernic, and C. Jacquemot. Synchronous programming with events and relations: the Signal language and its semantics. *Science of Computer Programming*, 16:103–149, 1991.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[DGH+99] J. Davis, M. Goel, C. Hylands, B. Kienhuis, E.A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong. Overview of the Ptolemy project. Technical Report UCB/ERL M99/37, University of California, Berkeley, 1999.

[DGV96] A. Deshpande, A. Gollu, and P. Varaiya. Shift: a formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems V*, Lecture Notes in Computer Science 1567. Springer-Verlag, 1996.

[Har87] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[LSVW96] N.A. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O Automata. In *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 496–510. Springer-Verlag, 1996.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[4] For the latest on Fresco activities, see `www.eecs.berkeley.edu/~fresco`.

A Single Complete Refinement Rule for Demonic Specifications

Karl Lermer[1] and Paul Strooper[2]

[1] Software Verification Research Centre
[2] Dept. of Comp. Science and Elec. Eng.,
The University of Queensland, Queensland 4072, Australia,
{lermer,pstroop}@csee.uq.edu.au

Abstract. We present the complete lattice of demonic languages and its interpretation in refinement proofs. In contrast to the conventional approach of refinement with an abstraction relation on the underlying state spaces, we introduce a notion of refinement with an abstraction relation on the power sets of the state spaces. This allows us to derive a single complete refinement rule for demonic specifications.

1 Introduction

In [10], a refinement semantics is presented for so-called demonic specifications that lifts the refinement of operations defined in formal specifiation languages such as Z and VDM to the refinement of state machines. The semantics is consistent with the conventional refinement semantics for the system underlying a specification in Z [14]. Rather than using a system or state machine semantics via the subset ordering of prefix-closed languages [5,6,14], an approach is taken where operations are interpreted as relations on state spaces, and input/output histories are used to define the semantics of the underlying state machine. This approach is not state dependent, as is the case for the improved failure model of CSP [4,6]. Instead, the languages and specifications are restricted to so-called demonic ones, where the enabling on a new input is dependent only on the past input history independent of the past outputs. The refinement relation is not simply trace inclusion where traces may disappear during the refinement process; it requires the refined system to accept the same or more traces as the original one. As usual, refinement relations on the operational level are defined via abstraction relations between the underlying state spaces. In analogy to standard simulation techniques [5,9,6,11,14,13], it is then possible to express every refinement with the help of two refinement methods on the operational level, denoted by *forward* and *backward* refinement.

A relational approach with abstraction relations as above will always necessitate two refinement rules for completeness. By using a predicate-transformer semantics, Gardiner and Morgan [2] obtain completeness of refinement by using a single refinement technique, called *cosimulation* (a predicate transformer with certain properties).

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 564–579, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In this paper, we build on the relational approach of [10] to further investigate the completeness of refinement. As refinement semantics we will use the partial ordering on *prefix-closed* and *demonic* languages introduced in [10]. We then extend the results from [10], by proving that this ordering defines a natural lattice structure on the prefix-closed and demonic languages, and discuss its importance for refinement proofs. This is in analogy to the subset ordering on languages and its interpretation in refinements [3,1,5,11,14,13]. Furthermore, we generalise the notion of refinement relations via abstraction relations on state spaces to refinement relations via abstraction relations on power sets of state spaces. As for the predicate transformer setting [2], it is then possible to derive a single complete refinement rule.

In Section 2 we introduce the notions of *prefix-closed* and *demonic* languages, and *demonic* specifications from [10]. In Section 3, we present the lattice structure on these languages, and in Section 4 we introduce a notion of refinement on specifications and show its soundness and completeness. We conclude in Section 5 with a comparison with related work.

2 Languages and Specifications

We view a module as a black box, accessible only through a fixed set of *operations* — the exported procedures and functions. A *module interface specification* (hereafter just *specification*) specifies the behaviour of the module. The *syntax* of the specification states the names of the access routines, and their inputs and outputs. We use Op to denote the set of all operation names, In to denote the set of all inputs, and Out to denote the set of all outputs.

The *semantics* of the specification describes the observable behaviour of the operations. We are interested in comparing the behaviour of different specifications. Because there are many ways to represent the state in a specification, we need a definition of behaviour that is independent of the state representation. We first consider *histories* (observable behaviours): finite, possibly empty sequences of the form

$$h = \langle c_1, v_1\rangle\langle c_2, v_2\rangle \ldots \langle c_n, v_n\rangle$$

For $i \in \{1, ..., n\}$, $c_i = \langle \iota_i, op_i\rangle$ is a call to an operation $op_i \in Op$ with input $\iota_i \in In$, and $v_i \in Out$ is an output. If an operation has no input or output, we use the special symbol $\perp$ to indicate this. We use the symbol ε to denote the empty history.

2.1 Languages

The set of all histories, $\mathcal{H}$, is determined by Op, In, and Out. A language $\mathcal{L}$ is defined as a subset of $\mathcal{H}$. We only consider non-empty languages that are *prefix-closed*: for any history $h \in \mathcal{L}$ and any call-value pair $\langle c, v\rangle$, if $h\langle c, v\rangle \in \mathcal{L}$ then $h \in \mathcal{L}$.

$$\mathcal{L}an_{\mathcal{H}} = \{\mathcal{L} \subseteq \mathcal{H} \mid \mathcal{L} \neq \varnothing \wedge \mathcal{L} \text{ is prefix-closed}\}$$

This implies $\varepsilon \in \mathcal{L}$ for all languages in $\mathcal{L}an_{\mathcal{H}}$. We introduce the following operators on histories. For any history

$$h = \langle c_1, v_1\rangle\langle c_2, v_2\rangle \ldots \langle c_n, v_n\rangle$$

we denote the corresponding *trace* or *input sequence* by

$$\mathcal{I}(h) = c_1\, c_2 \ldots c_n$$

As for histories, we use ε to denote the empty trace and thus $\mathcal{I}(\varepsilon) = \varepsilon$. For a set of histories $H \subseteq \mathcal{H}$, we define the set of all traces of H by

$$Tr(H) = \{\mathcal{I}(h) \mid h \in H\}$$

For a language $\mathcal{L}$ and a trace $t \in Tr(\mathcal{H})$, we collect all possible histories with trace t in the set

$$\Omega_{\mathcal{L}}(t) = \{h \mid h \in \mathcal{L} \wedge \mathcal{I}(h) = t\}$$

We will also use the following operators on finite sequences $\sigma = s_1 s_2 ... s_n$: $front(\sigma) = s_1 s_2 ... s_{n-1}$, $last(\sigma) = s_n$, and $\sigma \uparrow m = s_1 ... s_m$. Finally, we use $\#\sigma$ to denote the length of σ.

In [10], we show that the class of *demonic* languages provides a natural semantics for data refinement in VDM and Z. Intuitively, a language $\mathcal{L}$ is demonic if it is prefix-closed and the fact that an input is enabled depends solely on the past input history, independent of the corresponding outputs.

Definition 1. *A language $\mathcal{L}$ is demonic if*

a) $\mathcal{L}$ *is prefix-closed.*
b) $\forall\, h_1, h_2 \in \mathcal{L} : \mathcal{I}(h_1) = \mathcal{I}(h_2) \Rightarrow$
$\forall\, \iota \in In, \omega \in Out, op \in Op :$
$h_1\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L} \Rightarrow \exists\, \omega' \in Out : h_2\langle\langle \iota, op\rangle, \omega'\rangle \in \mathcal{L}$

The set of demonic languages will be denoted by

$$\mathcal{L}an^d_{\mathcal{H}} = \{\mathcal{L} \subseteq \mathcal{H} \mid \mathcal{L} \neq \varnothing \wedge \mathcal{L} \text{ is demonic}\}$$

For instance, deterministic languages ($\forall\, \tau \in Tr(\mathcal{L}) : \#\Omega_{\mathcal{L}}(\tau) = 1$) and total languages ($\forall\, h \in \mathcal{L}, \iota \in In, op \in Op\, \exists\, \omega \in Out : h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}$) are demonic.

A useful characterisation for demonic languages is the following: a language $\mathcal{L}$ is demonic iff

$$\forall\, \tau \in Tr(\mathcal{L}) \setminus \{\varepsilon\} \;:\; \Omega_{\mathcal{L}}(front(\tau)) \;=\; \{\, h \uparrow (\#\tau - 1) \mid h \in \Omega_{\mathcal{L}}(\tau)\,\}$$

Unfortunately, the set of demonic languages $\mathcal{L}an^d_{\mathcal{H}}$ does not behave as nicely as the set of prefix-closed languages $\mathcal{L}an_{\mathcal{H}}$, which forms a complete lattice under the inclusion ordering $\subseteq$ and the usual set operations. In general, demonic languages are not closed under intersection and union. We will see below that $\mathcal{L}an^d_{\mathcal{H}}$ carries a lattice structure under a different ordering relation.

We introduce a partial ordering $\Subset$ on languages. In Section 3, we use the poset (partially ordered set)

$$(\mathcal{L}an^d_{\mathcal{H}}, \Subset)$$

to define a lattice structure on $\mathcal{L}an^d_{\mathcal{H}}$, and in Section 4, we use this lattice as a domain for the characterisation of refinement proofs.

Definition 2. *Let $\mathcal{L}$ and $\mathcal{L}'$ be languages in $\mathcal{H}$,*

$$\mathcal{L}' \Subset \mathcal{L} \quad \textit{iff} \quad Tr(\mathcal{L}) \subseteq Tr(\mathcal{L}') \wedge \forall\, t \in Tr(\mathcal{L}) : \Omega_{\mathcal{L}'}(t) \subseteq \Omega_{\mathcal{L}}(t)$$

For languages $\mathcal{L}$ and $\mathcal{L}'$, $\mathcal{L}' \Subset \mathcal{L}$ if all traces in $\mathcal{L}$ also occur in $\mathcal{L}'$, and if every history in $\mathcal{L}'$ corresponding to a trace in $\mathcal{L}$ is also a history in $\mathcal{L}$.

2.2 Specifications

A specification S defines a language $\mathcal{L}$ — the subset of $\mathcal{H}$ expressing the behaviour defined by the specification. In general the form of the specification may vary, but in this paper we focus on *model-based* specifications, where the behaviour is specified in terms of a state space St.

Definition 3. *A (model-based) specification S is a six-tuple*

$$(Op, St, In, Out, \textsc{Init}, _^S)$$

with operation (name) set Op, state set St, input set In, output set Out, a nonempty set of initial states $\textsc{Init} \subseteq St$, and an interpretation function

$$_^S : Op \to \mathbb{P}\,((In \times St) \times (St \times Out))$$

Note that Op, St, In, Out, $\textsc{Init}$ can be infinite sets. Any operation $op \in Op$ is interpreted via $_^S$ as a set of pairs

$$(\langle \iota, s\rangle, \langle s', \omega\rangle)$$

where each pair represents a state transition with input $\iota \in In$, internal states $s, s' \in St$ (s denotes the state before and s' the state after the operation is performed), and output $\omega \in Out$.

For a specification S, the *precondition* of operation $op \in Op$ with input $\iota \in In$ will be denoted by

$$pre_S(\langle \iota, op\rangle) = \{ s \in St \mid \exists\, s' \in St, \omega \in Out : (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S \}$$

We recursively define the *postcondition* of a trace $t \in Tr(\mathcal{H})$ by

$$ptrace_S(t) = \begin{cases} \textsc{Init} & \text{if } t \text{ is the empty trace} \\ \{s' \in St \mid \exists\, s \in St, \omega \in Out : & \\ \qquad (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S \wedge s \in ptrace_S(t_1)\} & \\ \qquad \text{if } t = t_1\langle \iota, op\rangle & \end{cases}$$

Note that in our setting, pre- and postconditions denote sets of states, not predicates. Given a specification S and a history $h \in \mathcal{H}$, we denote the set of *final states* of h by

$$\mathit{final}_S(h) = \begin{cases} \mathit{INIT} & \text{if } h = \varepsilon \\ \{s' \in \mathit{St} \mid \exists\, s \in \mathit{St} \,:\, (\langle \iota, s\rangle, \langle s', \omega\rangle) \in \mathit{op}^S \wedge s \in \mathit{final}_S(h_1)\} & \\ & \text{if } h = h_1\langle\langle \iota, \mathit{op}\rangle, \omega\rangle \end{cases}$$

We can now define the language accepted by a specification S, consisting of the empty history and all histories that are produced by starting from an initial state in INIT and recursively applying the operations from Op.

Definition 4. *For a specification S, the language accepted by S is*

$$\mathcal{L}_S = \{h \in \mathcal{H} \mid h = \varepsilon \vee \exists\, h_1 \in \mathcal{L}_S, \mathit{op} \in \mathit{Op}, \iota \in \mathit{In}, \omega \in \mathit{Out} \,: \\ h = h_1\langle\langle \iota, \mathit{op}\rangle, \omega\rangle \wedge (\exists\, s \in \mathit{final}_S(h_1), s' \in \mathit{St} \,:\, (\langle \iota, s\rangle, \langle s', \omega\rangle) \in \mathit{op}^S)\}$$

It follows from this definition that $\mathcal{L}_S$ is prefix-closed (i.e., $\mathcal{L}_S \in \mathcal{L}an_{\mathcal{H}}$).

There is a notion corresponding to demonic languages for specifications. A specification S is demonic if whenever an input/operation pair $\langle \iota, \mathit{op}\rangle$ is enabled in a final state of a history $h \in \mathcal{L}_S$ it must be enabled in all the final states of the trace $\mathcal{I}(h)$. Again, the input enabling depends only on the input history.

Definition 5. *A specification S is demonic if*

$$\forall\, \tau \in \mathit{Tr}(\mathcal{L}_S) \setminus \{\varepsilon\} \;:\; \mathit{ptrace}_S(\mathit{front}(\tau)) \subseteq \mathit{pre}_S(\mathit{last}(\tau))$$

Every demonic specification S defines a demonic language $\mathcal{L}_S$ [10, Prop. 2]. The converse is not true in general: there are non-demonic specifications that specify demonic languages. However, for every demonic language $\mathcal{L}$ there exists a demonic specification that defines $\mathcal{L}$ [10, Prop. 11].

As an example of a demonic specification, consider the following random number generating specification $SA1$ from [10].

There are two operations: *random* generates a random integer value and has no output, and *val* returns the value generated by the last call to *random* as its output. If no call to *random* has been made, *val* returns 0.

$$\begin{aligned} \mathit{Op} &= \{\mathit{random}, \mathit{val}\}\,,\ \mathit{Out} = \mathbb{Z} \cup \{\bot\}\,,\ \mathit{In} = \{\bot\} \\ \mathit{St}^{SA1} &= \mathbb{Z}\,,\ \mathit{INIT}^{SA1} = \{0\} \\ \mathit{random}^{SA1} &= \{(\langle \bot, s\rangle, \langle s', \bot\rangle) \mid s, s' \in \mathbb{Z}\} \\ \mathit{val}^{SA1} &= \{(\langle \bot, i\rangle, \langle i, i\rangle) \mid i \in \mathbb{Z}\} \end{aligned}$$

By adding the operation $\mathit{two} = \{(\langle \bot, 2\rangle, \langle 2, 2\rangle)\}$ to specification $SA1$ we obtain a non-demonic specification. This is because *two* is only enabled at state 2 and not on all states that can be generated by *random*.

To motivate the use of demonic specifications and languages and to give a few generic examples, we state the correspondence to the commonly used failure

and trace models in the theory of communicating sequential processes (CSP) [6, 7].

Given a specification $S = (Op, St, In, Out, \mathit{INIT}, _^S)$ and a fresh symbol ζ we define two derived demonic specifications, the total completion

$$S(TC) = (Op, St \cup \{\zeta\}\, In, Out \cup \{\zeta\}, \mathit{INIT}, _^{S(TC)})$$

and the failure completion

$$S(FC) = (Op, St \cup \{\zeta\}, In, Out \cup \mathbb{P}(In \times Op \times Out), \mathit{INIT}, _^{S(FC)})$$

of S. For every operation symbol $op \in Op$ we define

$$\begin{aligned}
op^{S(TC)} &= op^S \cup \{(\langle \iota, s\rangle, \langle \zeta, \zeta\rangle) \,:\, \iota \in In \wedge s \in St \cup \{\zeta\}\} \\
op^{S(FC)} &= op^S \cup \{(\langle \iota, s\rangle, \langle \zeta, X\rangle) \,:\, \iota \in In \wedge s \in St \wedge X \in \mathbb{P}(In \times Op \times Out) \\
&\qquad \wedge\, \exists\, \omega \in Out \,:\, \langle\langle \iota, op\rangle, \omega\rangle \in X \\
&\qquad \wedge\, \forall \langle\langle \iota_0, op_0\rangle, \omega_0\rangle \in X \nexists s' \in St \,:\, (\langle \iota_0, s\rangle, \langle s', \omega_0\rangle) \in {op_0}^S\,\} \\
&\quad \cup \{(\langle \iota, \zeta\rangle, \langle \zeta, \varnothing\rangle) \,:\, \iota \in In\}
\end{aligned}$$

In both specifications the state ζ is interpreted as the state that the system enters after a failure occurred. The total completion extends the behaviour of the original specification by assuming that a failure can occur in any state, no matter what the input is: the system may enter the failure state at any moment. The failure completion handles failures in a more sophisticated manner. A failure transition can only occur if the transition was impossible in the original system. With the help of the set X the modified system can output failure transitions at any state.

The total and the failure completion of a specification S are demonic specifications.

Proposition 1. *For every specification S, $S(TC)$ and $S(FC)$ are demonic with total languages $\mathcal{L}_{S(TC)}$ and $\mathcal{L}_{S(FC)}$.*

Defining the failures of a specification S by

$$\begin{aligned}
\mathit{failures}(S) = \{(h, X) \,:\, & h \in \mathcal{L}_S \wedge X \in \mathbb{P}(In \times Op \times Out) \\
& \wedge\, \exists\, s \in \mathit{final}_S(h)\, \forall \langle\langle \iota, op\rangle, \omega\rangle \in X \nexists s' \in St \,:\, (\langle \iota, s\rangle, \langle s', \omega\rangle) \in op^S\}
\end{aligned}$$

we can formulate the following theorem. It shows that prominent failure and trace models of CSP [6,7] can be expressed via demonic specifications and the ordering relation $\Subset$. For brevity, and because the theorem is not used later on, the proofs have been omitted from the paper. The first equivalence is rather obvious and states that the inclusion relation on the languages (input/output trace sets) of the specifications is characterised by the $\Subset$ relation of the underlying total completions. The second equivalence characterises the inclusion relation on the failure sets of specifications in terms of the $\Subset$ relation on the underlying failure completions.

Theorem 1. *Let S^A and S^C be any specifications. Then, the following correspondences hold.*

i) $\mathcal{L}_{S^C(TC)} \Subset \mathcal{L}_{S^A(TC)} \Leftrightarrow \mathcal{L}_{S^C} \subseteq \mathcal{L}_{S^A}$
ii) $\mathcal{L}_{S^C(FC)} \Subset \mathcal{L}_{S^A(FC)} \Leftrightarrow failures(S^C) \subseteq failures(S^A)$

3 Lattice of Demonic Languages

In this section, we present the lattice structure that is induced on the prefix-closed and demonic languages by the ordering relation $\Subset$. First we introduce operators that define the supremum and infimum in the lattice. To obtain a complete lattice, we add the new symbol $\perp$ as the smallest element to both $\mathcal{L}an_{\mathcal{H}}$ and $\mathcal{L}an^d_{\mathcal{H}}$.

Definition 6. *i)* $\mathcal{L}_\varepsilon = \{\varepsilon\}$, $\mathcal{L}an_{\mathcal{H}}{}^* = \mathcal{L}an_{\mathcal{H}} \cup \{\perp\}$, $\mathcal{L}an^d_{\mathcal{H}}{}^* = \mathcal{L}an^d_{\mathcal{H}} \cup \{\perp\}$
ii) For a nonempty family of languages $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$, $i \in I$*:*

$$IT(\mathcal{L}_i) = \cap_{i\in I}\, Tr(\mathcal{L}_i)\,,\ UT(\mathcal{L}_i) = \cup_{i\in I}\, Tr(\mathcal{L}_i)$$

iii) For a nonempty family of languages $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$, $i \in I$*:*

$$\vee_I\, \mathcal{L}_i = \cup\{\Omega_{\mathcal{L}_i}(\tau) \mid i \in I \wedge \tau \in IT(\mathcal{L}_i)\}$$

$$\wedge_I\, \mathcal{L}_i = \begin{cases} \perp & \text{if } \exists \tau \in UT(\mathcal{L}_i) : \cap\{\Omega_{\mathcal{L}_i}(\tau) \mid i \in I \wedge \tau \in Tr(\mathcal{L}_i)\} = \varnothing \\ \cup\{\cap\{\Omega_{\mathcal{L}_i}(\tau) \mid i \in I \wedge \tau \in Tr(\mathcal{L}_i)\} \mid \tau \in UT(\mathcal{L}_i)\} & \text{otherwise} \end{cases}$$

$$\wedge^p_I\, \mathcal{L}_i = \begin{cases} \perp & \text{if } \nexists \mathcal{L} \in \mathcal{L}an_{\mathcal{H}} : \mathcal{L} \subseteq \wedge_I\, \mathcal{L}_i \wedge Tr(\mathcal{L}) = Tr(\wedge_I\, \mathcal{L}_i) \\ \cup\{\mathcal{L} \in \mathcal{L}an_{\mathcal{H}} \mid \mathcal{L} \subseteq \wedge_I\, \mathcal{L}_i \wedge Tr(\mathcal{L}) = Tr(\wedge_I\, \mathcal{L}_i)\} & \text{otherwise} \end{cases}$$

$$\wedge^d_I\, \mathcal{L}_i = \begin{cases} \perp & \text{if } \nexists \mathcal{L} \in \mathcal{L}an^d_{\mathcal{H}} : \mathcal{L} \subseteq \wedge_I\, \mathcal{L}_i \wedge Tr(\mathcal{L}) = Tr(\wedge_I\, \mathcal{L}_i) \\ \cup\{\mathcal{L} \in \mathcal{L}an^d_{\mathcal{H}} \mid \mathcal{L} \subseteq \wedge_I\, \mathcal{L}_i \wedge Tr(\mathcal{L}) = Tr(\wedge_I\, \mathcal{L}_i)\} & \text{otherwise} \end{cases}$$

It is possible that for prefix-closed languages $\mathcal{L}_i$, $i \in I$, the language $\wedge_I\, \mathcal{L}_i$ is not prefix-closed. In fact, we will see that $\wedge^p_I\, \mathcal{L}_i$, if not $\perp$, is the greatest prefix-closed language below $\wedge_I\, \mathcal{L}_i$ (w.r.t $\Subset$). Also, if all $\mathcal{L}_i$, $i \in I$ are demonic then $\wedge_I\, \mathcal{L}_i$ is not necessarily demonic. This is illustrated by the following example. Assume operations c_1 and c_2 with no inputs (we will use c_1 as a shorthand for the call $\langle \perp, c_1 \rangle$, and similarly for c_2), and outputs o_1, o_2, o_3, and o_4. We define the languages

$$\begin{aligned} \mathcal{L}_1 = \{&\varepsilon, \langle\langle c_1, o_1\rangle\rangle, \langle\langle c_1, o_2\rangle\rangle, \langle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_4\rangle\rangle, \\ &\langle\langle c_1, o_1\rangle\langle c_2, o_3\rangle\rangle, \langle\langle c_1, o_2\rangle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_3\rangle\langle c_1, o_2\rangle\rangle, \langle\langle c_2, o_4\rangle\langle c_1, o_2\rangle\rangle\} \\ \mathcal{L}_2 = \{&\varepsilon, \langle\langle c_1, o_1\rangle\rangle, \langle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_4\rangle\rangle, \langle\langle c_2, o_3\rangle\langle c_1, o_1\rangle\rangle, \langle\langle c_2, o_4\rangle\langle c_1, o_2\rangle\rangle\} \end{aligned}$$

$\mathcal{L}_1$, $\mathcal{L}_2$ are demonic languages with

$$\begin{aligned} \mathcal{L}_1 \wedge \mathcal{L}_2 = \{&\varepsilon, \langle\langle c_1, o_1\rangle\rangle, \langle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_4\rangle\rangle, \\ &\langle\langle c_1, o_1\rangle\langle c_2, o_3\rangle\rangle, \boxed{\langle\langle c_1, o_2\rangle\langle c_2, o_3\rangle\rangle}, \langle\langle c_2, o_4\rangle\langle c_1, o_2\rangle\rangle\} \\ \mathcal{L}_1 \wedge^p \mathcal{L}_2 = \{&\varepsilon, \langle\langle c_1, o_1\rangle\rangle, \boxed{\langle\langle c_2, o_3\rangle\rangle}, \langle\langle c_2, o_4\rangle\rangle, \end{aligned}$$

$$\langle\langle c_1, o_1\rangle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_4\rangle\langle c_1, o_2\rangle\rangle\}$$

$$\mathcal{L}_1 \wedge^d \mathcal{L}_2 = \{\varepsilon, \langle\langle c_1, o_1\rangle\rangle, \langle\langle c_2, o_4\rangle\rangle, \langle\langle c_1, o_1\rangle\langle c_2, o_3\rangle\rangle, \langle\langle c_2, o_4\rangle\langle c_1, o_2\rangle\rangle\}$$

The following two lemmas prove properties of $\wedge_I \mathcal{L}_i$, $\wedge_I^p \mathcal{L}_i$, $\wedge_I^d \mathcal{L}_i$, and $\vee_I \mathcal{L}_i$ that will allow us to derive the complete lattice structure for languages ordered by $\Subset$.

Lemma 1 *For any nonempty family of languages $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$, $i \in I$:*
a) $\wedge_I^p \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}{}^*$ and $\wedge_I^d \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}^{d\,*}$
b) If $\wedge_I \mathcal{L}_i \neq \perp$ then $\wedge_I \mathcal{L}_i \Subset \mathcal{L}_j$ for all $j \in I$

Proof. a) The union of prefix-closed languages is again prefix-closed, and the union of demonic languages with the same set of traces is also demonic [10, Prop. 1].

b) For any $j \in I$, $Tr(\mathcal{L}_j) \subseteq UT(\mathcal{L}_i) = Tr(\wedge_I \mathcal{L}_i)$ and for $\tau \in Tr(\mathcal{L}_j)$ we get $\Omega_{\wedge_I \mathcal{L}_i}(\tau) = \cap\{\Omega_{\mathcal{L}_i}(\tau) \mid i \in I \wedge \tau \in Tr(\mathcal{L}_i)\} \subseteq \Omega_{\mathcal{L}_j}(\tau)$ □

Lemma 2 *For any nonempty family of languages $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$, $i \in I$:*
a) $\vee_I \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$
b) If all $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}^d$, then $\vee_I \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}^d$

Proof. a) Let $hz \in \vee_I \mathcal{L}_i$. Then, $\mathcal{I}(hz) \in IT(\mathcal{L}_i)$ and since $\mathcal{L}_i$, $i \in I$, is prefix-closed, we can conclude $\mathcal{I}(h) \in IT(\mathcal{L}_i)$. Moreover, for all $i_0 \in I$ for which $hz \in \mathcal{L}_{i_0}$, because $\mathcal{L}_{i_0}$ is prefix-closed, we have $h \in \mathcal{L}_{i_0}$ and so, by definition of the $\vee$-operator, $h \in \vee_I \mathcal{L}_i$.

b) Let $\mathcal{L}_i$, $i \in I$ be demonic. Then we fix a trace $\tau \in Tr(\vee_I \mathcal{L}_i) \setminus \{\varepsilon\}$ and some $h \in \Omega_{\vee_I \mathcal{L}_i}(front(\tau))$. Because

$$\Omega_{\vee_I \mathcal{L}_i}(front(\tau)) = \cup\{\Omega_{\mathcal{L}_i}(front(\tau)) \mid i \in I\}$$

we can find $i_0 \in I$ with $h \in \Omega_{\mathcal{L}_{i_0}}(front(\tau))$. Since $\mathcal{L}_{i_0}$ is demonic and $\tau \in \cap_{i \in I} Tr(\mathcal{L}_i)$ there exists a $h' \in \Omega_{\mathcal{L}_{i_0}}(\tau)$ with $h' \uparrow (\#\tau - 1) = h$. We can conclude $h' \in \Omega_{\vee_I \mathcal{L}_i}(\tau)$ and so $\vee_I \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}^d$. □

We also need the following result, proven in [10, Prop. 5].

Lemma 3 *For languages $\mathcal{L}, \mathcal{L}' \subseteq \mathcal{H}$ we have:*
a) $\mathcal{L}' \Subset \mathcal{L} \Rightarrow Tr(\mathcal{L} \cap \mathcal{L}') = Tr(\mathcal{L}) \cap Tr(\mathcal{L}') = Tr(\mathcal{L})$, $\mathcal{L}' \Subset \mathcal{L}' \cap \mathcal{L}$
b) ($\mathcal{L}'$ demonic $\wedge$ $\mathcal{L}' \Subset \mathcal{L}$) $\Rightarrow$ $\mathcal{L}' \cap \mathcal{L}$ demonic

If we extend the ordering $\Subset$ in a canonical way from $\mathcal{L}an_{\mathcal{H}}$ to $\mathcal{L}an_{\mathcal{H}}{}^*$ such that $\perp$ becomes the smallest element in $\mathcal{L}an_{\mathcal{H}}{}^*$, then we do obtain a complete lattice. Furthermore, we set $\vee_\varnothing \mathcal{L}_i = \perp$ and $\wedge_\varnothing \mathcal{L}_i = \wedge_\varnothing^p \mathcal{L}_i = \wedge_\varnothing^d \mathcal{L}_i = \mathcal{L}_\varepsilon$.

Theorem 2. *a) $(\mathcal{L}an_{\mathcal{H}}{}^*, \Subset)$ is a complete lattice with greatest element $\mathcal{L}_\varepsilon$ and smallest element $\perp$. For a family $\mathcal{L}_i$, $i \in I$ of languages in $\mathcal{L}an_{\mathcal{H}}$, its supremum is $\vee_{\{i \in I \mid \mathcal{L}_i \neq \perp\}} \mathcal{L}_i$ and its infimum is $\perp$ if there is at least one $i \in I$ with $\mathcal{L}_i = \perp$ and $\wedge_I^p \mathcal{L}_i$ otherwise.*

b) $(\mathcal{L}an_{\mathcal{H}}^{d\,*}, \Subset)$ *is a complete lattice with greatest element* $\mathcal{L}_\varepsilon$ *and smallest element* $\perp$*. For a family* $\mathcal{L}_i$*,* $i \in I$ *of languages in* $\mathcal{L}an_{\mathcal{H}}^{d}$*, its supremum is* $\vee_{\{i\in I \mid \mathcal{L}_i \neq \perp\}}\ \mathcal{L}_i$ *and its infimum is* $\perp$ *if there is at least one* $i \in I$ *with* $\mathcal{L}_i = \perp$ *and* $\wedge_I^d\ \mathcal{L}_i$ *otherwise.*

Proof. a) Let $\mathcal{L} \in \mathcal{L}an_{\mathcal{H}}$. Then, $\{\varepsilon\} = Tr(\mathcal{L}_\varepsilon) \subseteq Tr(\mathcal{L})$ and $\Omega_{\mathcal{L}}(\varepsilon) = \{\varepsilon\} \subseteq \Omega_{\mathcal{L}_\varepsilon}(\varepsilon)$. Hence, $\mathcal{L} \Subset \mathcal{L}_\varepsilon$ which shows that $\mathcal{L}_\varepsilon$ is the greatest element in $\mathcal{L}an_{\mathcal{H}}{}^*$.

Let $\mathcal{L}_i$, $i \in I$ be a nonempty family of languages in $\mathcal{L}an_{\mathcal{H}}$. Applying Lemma 2 a) shows $\vee_I\ \mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$. For any $j \in I$, $Tr(\vee_I\ \mathcal{L}_i) = IT(\mathcal{L}_i) \subseteq Tr(\mathcal{L}_j)$. For any trace $\tau \in Tr(\vee_I\ \mathcal{L}_i)$,

$$\Omega_{\mathcal{L}_j}(\tau) \subseteq \cup_{i\in I}\Omega_{\mathcal{L}_i}(\tau) = \Omega_{\vee_I \mathcal{L}_i}(\tau)$$

Therefore, $\mathcal{L}_j \Subset \vee_I\ \mathcal{L}_i$, $j \in I$.

Moreover, for any $\mathcal{L} \in \mathcal{L}an_{\mathcal{H}}$ with $\mathcal{L}_i \Subset \mathcal{L}$, we have $Tr(\mathcal{L}) \subseteq IT(\mathcal{L}_i) = Tr(\vee_I\ \mathcal{L}_i)$ and for $\tau \in Tr(\mathcal{L})$ we have

$$\Omega_{\vee_I \mathcal{L}_i}(\tau) = \cup_{i\in I}\Omega_{\mathcal{L}_i}(\tau) \subseteq \cup_{i\in I}\Omega_{\mathcal{L}}(\tau) = \Omega_{\mathcal{L}}(\tau)$$

We deduce $\vee_I\ \mathcal{L}_i \Subset \mathcal{L}$.

This proves that $\vee_I\ \mathcal{L}_i$ is the least upper bound of the family $\mathcal{L}_i$, $i \in I$ in $\mathcal{L}an_{\mathcal{H}}$. It remains to extend this in a straightforward manner to $\mathcal{L}an_{\mathcal{H}}{}^*$.

Next we claim $\wedge_I\ \mathcal{L}_i \Subset \mathcal{L}_j$ for every $j \in I$. This is trivial if $\wedge_I\ \mathcal{L}_i = \perp$ and the remaining case has been shown in Lemma 1 b).

Therefore, $\wedge_I^p\ \mathcal{L}_i \subseteq \wedge_I\ \mathcal{L}_i \Subset \mathcal{L}_j$ for every $j \in I$. Since $\wedge_I^p\ \mathcal{L}_i$, if not equal to $\perp$, has the same set of traces as $\wedge_I\ \mathcal{L}_i$, we can infer

$$\wedge_I^p\ \mathcal{L}_i \Subset \wedge_I\ \mathcal{L}_i \Subset \mathcal{L}_j$$

for every $j \in I$. Let $\mathcal{L} \in \mathcal{L}an_{\mathcal{H}}{}^*$ with $\mathcal{L} \Subset \mathcal{L}_j$, for all $j \in I$. We must prove that $\mathcal{L} \Subset \wedge_I^p\ \mathcal{L}_i$, which trivially holds for $\mathcal{L} = \perp$. For $\mathcal{L} \neq \perp$, we first prove that $\mathcal{L} \Subset \wedge_I\ \mathcal{L}_i$. Note that $\varnothing \neq Tr(\wedge_I\ \mathcal{L}_i) = UT(\mathcal{L}_i) \subseteq Tr(\mathcal{L})$, Moreover, $Tr(\mathcal{L}_j) \subseteq Tr(\mathcal{L})$ for all $j \in I$ and, if $\tau \in UT(\mathcal{L}_i)$ then

$$\Omega_{\mathcal{L}}(\tau) \subseteq \cap\{\Omega_{\mathcal{L}_i}(\tau) \mid i \in I \wedge \tau \in Tr(\mathcal{L}_i)\} = \Omega_{\wedge_I \mathcal{L}_i}(\tau)$$

Hence, $\mathcal{L} \Subset \wedge_I\ \mathcal{L}_i$.

We define $\mathcal{L}' = \mathcal{L} \cap (\wedge_I\ \mathcal{L}_i)$ and claim the following two properties.

$$\mathcal{L}' \in \mathcal{L}an_{\mathcal{H}} \tag{1}$$

$$\mathcal{L} \Subset \mathcal{L}' \Subset \wedge_I^p\ \mathcal{L}_i \Subset \wedge_I\ \mathcal{L}_i \tag{2}$$

We show (1) first. Note that $\varepsilon \in \mathcal{L}'$. Let $hz \in \mathcal{L}'$. $\mathcal{L}$ is prefix-closed and so $h \in \mathcal{L}$. Furthermore, $\mathcal{I}(hz) \in Tr(\wedge_I\ \mathcal{L}_i) = UT(\mathcal{L}_i)$. From $\mathcal{L}_i$, $i \in I$ being prefix-closed, we can deduce $\mathcal{I}(h) \in Tr(\wedge_I\ \mathcal{L}_i)$. Since $\mathcal{L} \Subset \wedge_I\ \mathcal{L}_i$, we can conclude $h \Subset \wedge_I\ \mathcal{L}_i$ and therefore $h \in \mathcal{L}'$.

To prove (2), we note that $\wedge_I^p\ \mathcal{L}_i \Subset \wedge_I\ \mathcal{L}_i$, which was shown above. $\mathcal{L}'$ is prefix-closed by (1), and furthermore $\mathcal{L}' \subseteq \wedge_I\ \mathcal{L}_i$ and $\mathcal{L} \Subset \wedge_I\ \mathcal{L}_i$. By Lemma 3 a),

we can conclude $Tr(\mathcal{L}') = Tr(\wedge_I \mathcal{L}_i)$. Hence, by definition of the $\wedge^p$-operator we obtain $\mathcal{L}' \subseteq \wedge_I^p \mathcal{L}_i$. That $\mathcal{L}'$ and $\wedge_I^p \mathcal{L}_i$ have the same traces then implies

$$\mathcal{L}' \Subset \wedge_I^p \mathcal{L}_i$$

The property $\mathcal{L} \Subset \mathcal{L}'$ follows from Lemma 3 a) and $\mathcal{L} \Subset \wedge_I \mathcal{L}_i$.

From (2) we obtain $\mathcal{L} \Subset \wedge_I^p \mathcal{L}_i$ by transitivity of $\Subset$, which is what we wanted to verify.

b) It is obvious that $\mathcal{L}_\varepsilon$ is demonic and since $\mathcal{L}_\varepsilon$ is the greatest element in $\mathcal{L}an_{\mathcal{H}}{}^*$, it follows that $\mathcal{L}_\varepsilon$ is also the greatest element in $\mathcal{L}an_{\mathcal{H}}^{d\,*}$.

For any family $\mathcal{L}_i$, $i \in I$ in $\mathcal{L}an_{\mathcal{H}}^d$ we get $\vee_I \mathcal{L}_i$ as its supremum in $\mathcal{L}an_{\mathcal{H}}$. According to Lemma 2 b), $\vee_I \mathcal{L}_i$ is demonic and therefore, it is the supremum of $\mathcal{L}_i$, $i \in I$ in $\mathcal{L}an_{\mathcal{H}}^d$.

It remains to show that $\wedge_I^d \mathcal{L}_i$ is the greatest lower bound for $\mathcal{L}_i$, $i \in I$ in $\mathcal{L}an_{\mathcal{H}}^d$. From the definitions of $\wedge_I^p \mathcal{L}_i$ and $\wedge_I^d \mathcal{L}_i$ we can infer $\wedge_I^d \mathcal{L}_i \Subset \wedge_I^p \mathcal{L}_i$ and together with a) we then obtain

$$\wedge_I^d \mathcal{L}_i \Subset \wedge_I^p \mathcal{L}_i \Subset \mathcal{L}_j \,,\; j \in I$$

Now, assume $\mathcal{L} \in \mathcal{L}an_{\mathcal{H}}^d$ with $\mathcal{L} \Subset \mathcal{L}_i$, for all $i \in I$. With a) we deduce

$$\mathcal{L} \Subset \wedge_I^p \mathcal{L}_i \Subset \wedge_I \mathcal{L}_i$$

As above, by setting $\mathcal{L}' = \mathcal{L} \cap (\wedge_I \mathcal{L}_i)$ and by using Lemma 3 b), we obtain that $\mathcal{L}'$ is demonic. Property (2) holds again and by definition of $\wedge_I^d \mathcal{L}_i$,

$$\mathcal{L}' \subseteq \wedge_I^d \mathcal{L}_i$$

Hence, $\mathcal{L} \Subset \mathcal{L}' \subseteq \wedge_I^d \mathcal{L}_i$. Note that $Tr(\wedge_I^d \mathcal{L}_i) = Tr(\wedge_I \mathcal{L}_i) = Tr(\mathcal{L}')$ which follows from Lemma 3 a). Therefore,

$$\mathcal{L} \Subset \mathcal{L}' \Subset \wedge_I^d \mathcal{L}_i$$

and from the transitivity of $\Subset$ we can conclude that $\mathcal{L} \Subset \wedge_I^d \mathcal{L}_i$. □

The following results further explore the correspondence between the operators $\wedge$, $\wedge^p$ and $\wedge^d$.

Proposition 1 *If $\mathcal{L}_i \in \mathcal{L}an_{\mathcal{H}}$, $i \in I$, is a downward ordered net (i.e., $\forall\, i, j \in I\ \exists\, k \in I\ :\ \mathcal{L}_k \Subset \mathcal{L}_i \wedge \mathcal{L}_k \Subset \mathcal{L}_j$), then $\wedge_I \mathcal{L}_i = \wedge_I^p \mathcal{L}_i$.*

Proof. For a downward-ordered net $\mathcal{L}_i$, $i \in I$, since $\wedge_I^p \mathcal{L}_i$, if not equal to $\bot$, is the greatest prefix-closed set in $\wedge_I \mathcal{L}_i$, we simply have to show that $\wedge_I \mathcal{L}_i$ is prefix-closed.

Let $hz \in \wedge_I \mathcal{L}_i$. There is $i_0 \in I$ such that $hz \in \mathcal{L}_{i_0}$. $\mathcal{L}_{i_0}$ is prefix-closed, and so $h \in \mathcal{L}_{i_0}$. Thus, $\mathcal{I}(h) \in Tr(\wedge_I \mathcal{L}_i)$.

Now for all $i_1 \in I$ such that $\mathcal{I}(h) \in Tr(\mathcal{L}_{i_1})$, there exists an $i_2 \in I$ with $\mathcal{L}_{i_2} \Subset \mathcal{L}_{i_0}$ and $\mathcal{L}_{i_2} \Subset \mathcal{L}_{i_1}$. Hence, $\mathcal{I}(hz) \in Tr(\mathcal{L}_{i_2})$ and $hz \in \mathcal{L}_{i_2}$. $\mathcal{L}_{i_2}$ is prefix-closed and so $h \in \mathcal{L}_{i_2}$. From $\Omega_{\mathcal{L}_{i_2}}(\mathcal{I}(h)) \subseteq \Omega_{\mathcal{L}_{i_1}}(\mathcal{I}(h))$ we conclude $h \in \mathcal{L}_{i_1}$.

From this we conclude $h \in \wedge_I \mathcal{L}_i$, which proves that $\wedge_I \mathcal{L}_i$ is prefix-closed. □

Definition 7. *A language $\mathcal{L}$ is τ-output-finite for a trace $\tau \in Tr(\mathcal{L}) \setminus \{\varepsilon\}$ if for all histories $h \in \Omega_{\mathcal{L}}(front(\tau))$ the set*

$$\{h' \mid h' \in \Omega_{\mathcal{L}}(\tau) \wedge h' \uparrow \#h = h\}$$

is finite. $\mathcal{L}$ is output-finite if it is τ-output-finite for all $\tau \in Tr(\mathcal{L}) \setminus \{\varepsilon\}$.

Proposition 2 *If $\mathcal{L}_i \in \mathcal{L}an^d_{\mathcal{H}}$, $i \in I$ is a downward-ordered net and for all $\tau \in Tr(\wedge_I \mathcal{L}_i)$ there exists an $i \in I$ with $\mathcal{L}_i$ being τ-output-finite, then $\wedge_I \mathcal{L}_i = \wedge^d_I \mathcal{L}_i$.*

Proof. It suffices to prove that $\wedge_I \mathcal{L}_i$ is demonic, assuming $\wedge_I \mathcal{L}_i \neq \bot$. Let $\tau \in Tr(\wedge_I \mathcal{L}_i) \setminus \{\varepsilon\}$ and $h \in \Omega_{\wedge_I \mathcal{L}_i}(front(\tau))$. We find $i_1 \in I$ with $\mathcal{L}_{i_1}$ being τ-output finite. Furthermore, $\mathcal{L}_{i_1}$ is demonic, and so we can conclude $front(\tau) \in Tr(\mathcal{L}_{i_1})$ and that the set

$$\mathcal{A} = \{h' \mid h' \in \Omega_{\mathcal{L}_{i_1}}(\tau) \wedge h' \uparrow \#h = h\}$$

is nonempty and finite. We claim that there exists an $h' \in \mathcal{A}$ with $h' \in \wedge_I \mathcal{L}_i$. If not, we can find $j_1, ..., j_n \in I$ such that $\tau \in \cap_{k=1}^n Tr(\mathcal{L}_{j_k})$ and $\cap_{k=1}^n \Omega_{\mathcal{L}_{j_k}}(\tau) \cap \mathcal{A} = \varnothing$. But there exists an $i_2 \in I$ with $\mathcal{L}_{i_2} \Subset \mathcal{L}_{i_1}$ and $\mathcal{L}_{i_2} \Subset \mathcal{L}_{j_k}$, $1 \leq k \leq n$. We infer $\tau \in Tr(\mathcal{L}_{i_2})$ and $h \in \mathcal{L}_{i_2}$. Again, $\mathcal{L}_{i_2}$ is demonic and so

$$\varnothing \neq \{h' \mid h' \in \Omega_{\mathcal{L}_{i_2}}(\tau) \wedge h' \uparrow \#h = h\} \subseteq \cap_{k=1}^n \Omega_{\mathcal{L}_{j_k}}(\tau) \cap \mathcal{A}$$

which contradicts the assumption above. □

This shows that $\wedge_I \mathcal{L}_i$ is demonic, and hence $\wedge_I \mathcal{L}_i = \wedge^d_I \mathcal{L}_i$, for downward-ordered nets of output-finite and demonic languages $\mathcal{L}_i$, $i \in I$.

4 Refinement

Data-refinement proofs [3,1,8,14,13] are used to verify that a specification (or implementation) S^C with a concrete state representation is correct with respect to a specification S^A with an abstract state representation. In [10], we show the soundness and completeness of refinement with respect to the ordering $\Subset$ on demonic languages using a combination of forward and backward refinement. Here we generalise this notion of refinement to one that is based on an abstraction relation on the power sets of the underlying state spaces, and show that this provides a single sound and complete refinement rule.

Definition 8. *Given two specifications $S^A = (Op, St^A, In, Out, \textsc{Init}^A, _^{S^A})$ and $S^C = (Op, St^C, In, Out, \textsc{Init}^C, _^{S^C})$, an abstraction relation*

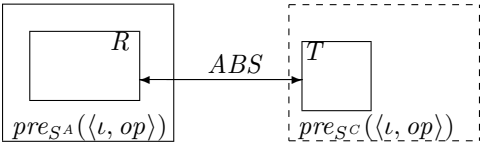
$$ABS : \mathbb{P}\, St^C \leftrightarrow \mathbb{P}\, St^A$$

and operation $op \in Op$, we say that op^{S^A} power-refines to op^{S^C} ($op^{S^A} \sqsubseteq_{ABS} op^{S^C}$) if the following conditions hold.

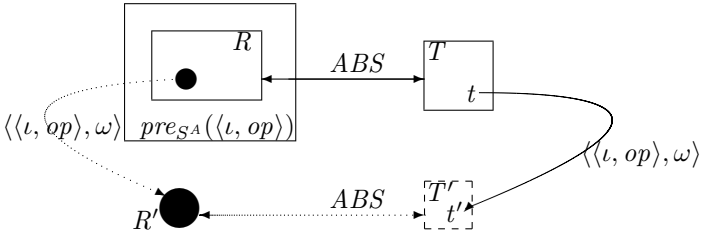
$(P1)$ $\forall \iota \in In\,,\ R \in \mathbb{P}\, St^A\,,\ T \in \mathbb{P}\, St^C\ :$
$((T,R) \in ABS \wedge \varnothing \neq R \subseteq pre_{S^A}(\langle \iota, op\rangle)) \Rightarrow\ T \subseteq pre_{S^C}(\langle \iota, op\rangle)$

$(P2)$ $\forall \iota \in In\,, \omega \in Out\,,\ R \in \mathbb{P}\, St^A\,,\ T \in \mathbb{P}\, St^C\,,\ t \in T, t' \in St^C\ :$
$((T,R) \in ABS \wedge \varnothing \neq R \subseteq pre_{S^A}(\langle \iota, op\rangle) \wedge (\langle \iota, t\rangle, \langle t', \omega\rangle) \in op^{S^C}) \Rightarrow$
$(\exists\, R' \in \mathbb{P}\, St^A\,,\ T' \in \mathbb{P}\, St^C\ :$
$(R' \neq \varnothing \wedge (T', R') \in ABS \wedge t' \in T'$
$\wedge\ \ (\forall\, s_1 \in R'\, \exists\, s_0 \in R\ :\ (\langle \iota, s_0\rangle, \langle s_1, \omega\rangle) \in op^{S^A}))$

Refinement with properties $(P1)$ and $(P2)$ lifts the common forward refinement of VDM and Z [8,10,14] with abstraction relations on the state spaces to refinement with abstraction relations on power sets. $(P1)$ states that if T and R are related via ABS and $\langle \iota, op\rangle$ is enabled in all states in R then $\langle \iota, op\rangle$ is enabled in all states in T (denoted by the dashed lines in the picture below).



$(P2)$ states that via ABS every input of op^{S^A} must be accepted by op^{S^C} with outputs that were possible for op^{S^A}. The last condition in $(P2)$ asserts that all states that are contained in the abstract state set R' and related to a concrete state set T' are final states under the abstract operation with input and output that were accepted by the concrete operation. This is illustrated in the picture below where the dashed lines indicate how the concrete transition is simulated by the abstract one.



This notion of operation refinement naturally leads to the following notion of specification refinement.

Definition 9. *We say that specification* $S^A = (Op, St^A, In, Out, \textit{INIT}^A, _^{S^A})$ *power-refines to specification* $S^C = (Op, St^C, In, Out, \textit{INIT}^C, _^{S^C})$ *and write* $S^A \sqsubseteq S^C$ *if there exists an abstraction relation* ABS *as above such that*

(PR1) $\forall\, op \in Op\ :\ op^{S^A} \sqsubseteq_{ABS} op^{S^C}$
(PR2) $(\textit{INIT}^C, \textit{INIT}^A) \in ABS$

We write $S^A \sqsubseteq_{ABS} S^C$ *if we want to explicitly indicate the abstraction relation* ABS.

We will see below that the relation $\sqsubseteq$ defines a preorder on demonic specifications. Obligation $(PR1)$ states that every abstract operation can be power-refined to a concrete one and obligation $(PR2)$ states that the abstract initial state set corresponds to the concrete initital state set. Note that we overload the semantics of the symbol $\sqsubseteq$. It will be obvious from the context whether we mean operation or specification refinement.

As an example, consider the following alternative specification $SA2$ of the random number generating specification $SA1$ of Section 2.2.

$$\begin{aligned} St^{SA2} &= \mathbb{Z} \cup \{undef\}\,,\ \mathit{INIT}^{SA2} = \{0\} \\ random^{SA2} &= \{(\langle \bot, s_0\rangle, \langle undef, \bot\rangle) \mid s_0 \in \mathbb{Z} \cup \{undef\}\} \\ val^{SA2} &= \{(\langle \bot, i\rangle, \langle i, i\rangle) \mid i \in \mathbb{Z}\} \cup \{(\langle \bot, undef\rangle, \langle i, i\rangle) \mid i \in \mathbb{Z}\} \end{aligned}$$

In specification $SA1$ the operation *random* generates a random integer number and a subsequent call to *val* makes this number visible. In $SA2$, *random* will always set the state to *undef* and only a subsequent call to *val* will generate a random number. In [10], we show that there is a backward refinement from $SA1$ to $SA2$, but no forward refinement. However, there is a power-refinement such that $SA1$ power-refines to $SA2$. An abstraction relation that shows this is

$$ABS = \{(\{i\}, \{i\}) : i \in \mathbb{Z}\} \cup \{(\{undef\}, \mathbb{Z})\}$$

If we use the ordering $\Subset$ as the underlying semantics of specification refinement and power-refinement with obligations $(PR1)$ and $(PR2)$ as the refinement technique, then Theorem 3 below states the soundness and completeness of power-refinement for demonic specifications.

Theorem 3. *For demonic specifications* S^A *and* S^C, $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A} \Leftrightarrow S^A \sqsubseteq S^C$.

Proof. To prove $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A} \Rightarrow S^A \sqsubseteq S^C$, we use the abstraction relation

$$(T, R) \in ABS \text{ iff } \exists\, h \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A} : T = \mathit{final}_{S^C}(h) \wedge R = \mathit{final}_{S^A}(h)$$

Because $\varepsilon \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A}$, $\mathit{INIT}^C = \mathit{final}_{S^C}(\varepsilon)$, and $\mathit{INIT}^A = \mathit{final}_{S^A}(\varepsilon)$, we can conclude $(\mathit{INIT}^C, \mathit{INIT}^A) \in ABS$, which establishes $(PR2)$.

For $(P1)$, we assume $(T, R) \in ABS$ and $\varnothing \neq R \subseteq \mathit{pre}_{S^A}(\langle \iota, op\rangle)$. According to the definition of ABS, there is $h \in \mathcal{L}_{S^C} \cap \mathcal{L}_{S^A}$ with $T = \mathit{final}_{S^C}(h)$ and $R = \mathit{final}_{S^A}(h)$. From $\mathcal{I}(h)\langle \iota, op\rangle \in \mathit{Tr}(\mathcal{L}_{S^A})$ and $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$, it follows $\mathcal{I}(h)\langle \iota, op\rangle \in \mathit{Tr}(\mathcal{L}_{S^C})$. S^C is demonic and therefore $T = \mathit{final}_{S^C}(h) \subseteq \mathit{ptrace}_{S^C}(\mathcal{I}(h)) \subseteq \mathit{pre}_{S^C}(\langle \iota, op\rangle)$.

To prove $(P2)$, we additionally assume $\omega \in \mathit{Out}$, $t \in T$ and $t' \in St^C$ with $(\langle \iota, t\rangle, \langle t', \omega\rangle) \in op^{S^C}$. Hence, $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^C}$ and because of $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$ we have $h\langle\langle \iota, op\rangle, \omega\rangle \in \mathcal{L}_{S^A}$. For $T' = \mathit{final}_{S^C}(h\langle\langle \iota, op\rangle, \omega\rangle)$ and $R' = \mathit{final}_{S^A}(h\langle\langle \iota, op\rangle, \omega\rangle)$ we derive $(T', R') \in ABS$, $t' \in T'$ and $R' \neq \varnothing$. Finally,

because $R' = \mathit{final}_{S^A}(h\langle\langle \iota, op\rangle, \omega\rangle)$, for any $s_1 \in R'$ there is a $s_0 \in R$ such that $(\langle \iota, s_0\rangle, \langle s_1, \omega\rangle) \in {op^{S^A}}$, which concludes the proof of $(P2)$.

To prove $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A} \Leftarrow S^A \sqsubseteq S^C$, we first prove the following property by induction on the length of the traces in $Tr(\mathcal{L}_{S^A})$.

$$
\begin{array}{l}
\forall\, \tau = \langle \iota_1, op_1\rangle ... \langle \iota_{\#\tau}, op_{\#\tau}\rangle \in Tr(\mathcal{L}_{S^A}) \\
\quad \exists\ R_0, ..., R_{\#\tau}, T_0, ..., T_{\#\tau}, t_0, ..., t_{\#\tau}, \omega_1, ..., \omega_{\#\tau}\ : \\
\quad a)\ R_0 = \mathit{INIT}^A \wedge T_0 = \mathit{INIT}^C \\
\quad b)\ \forall\ 0 \leq i \leq \#\tau\ :\ t_i \in T_i \wedge (T_i, R_i) \in ABS \\
\quad c)\ \forall\ 1 \leq i \leq \#\tau\ :\ (\langle \iota_i, t_{i-1}\rangle, \langle t_i, \omega_i\rangle) \in {op_i}^{S^C} \\
\quad d)\ \forall\, 0 \leq i \leq \#\tau\ :\ \varnothing \neq R_i \subseteq \mathit{ptrace}_{S^A}(\tau \uparrow i) \\
\quad e)\ \forall\, 1 \leq i \leq \#\tau\, \forall\, s_1 \in R_i\, \exists\, s_0 \in R_{i-1}\ :\ (\langle \iota_i, s_0\rangle, \langle s_1, \omega_i\rangle) \in {op_i}^{S^A}
\end{array}
\tag{3}
$$

Base case ($\tau = \varepsilon$): We have $(\mathit{INIT}^C, \mathit{INIT}^A) \in ABS$ and $\mathit{ptrace}_{S^A}(\varepsilon) = \mathit{INIT}^A \neq \varnothing$ by definition. Similarly, we can select $t_0 \in T_0 = \mathit{INIT}^C \neq \varnothing$.

Induction step ($\tau' = \tau\langle \iota, op\rangle \in Tr(\mathcal{L}_{S^A})$): For $\tau \in Tr(\mathcal{L}_{S^A})$ we can find $R_0, ..., R_{\#\tau}$, $T_0, ..., T_{\#\tau}, t_0, ..., t_{\#\tau}, \omega_1, ..., \omega_{\#\tau}$ that satisfy property (3) according to our induction hypothesis. Since S^A is demonic, it follows from property (3) d) that $\varnothing \neq R_{\#\tau} \subseteq \mathit{ptrace}_{S^A}(\tau) \subseteq \mathit{pre}_{S^A}(\langle \iota, op\rangle)$. From $(P1)$ and $(T_{\#\tau}, R_{\#\tau}) \in ABS$ we can infer $T_{\#\tau} \subseteq \mathit{pre}_{S^C}(\langle \iota, op\rangle)$. Then we find $t' \in St^C$, $\omega \in \mathit{Out}$ such that $(\langle \iota, t_{\#\tau}\rangle, \langle t', \omega\rangle) \in {op^{S^C}}$. Property $(P2)$ then supplies us with $R' \in \mathbb{P}\, St^A$, $T' \in \mathbb{P}\, St^C$ such that $R' \neq \varnothing$, $(T', R') \in ABS$ and $t' \in T'$. Furthermore, for any $s_1 \in R'$ there is a $s_0 \in R_{\#\tau}$ with $(\langle \iota, s_0\rangle, \langle s_1, \omega\rangle) \in {op^{S^A}}$. We know by our induction hypothesis that $R_{\#\tau} \subseteq \mathit{ptrace}_{S^A}(\tau)$ and therefore $R' \subseteq \mathit{ptrace}_{S^A}(\tau')$. The induction step is completed by setting $R_{\#\tau'} = R'$, $T_{\#\tau'} = T'$, $t_{\#\tau'} = t'$ and $\omega_{\#\tau'} = \omega$.

We proceed by proving the following property by induction on the length of the traces in $Tr(\mathcal{L}_{S^C}) \cap Tr(\mathcal{L}_{S^A})$.

$$
\begin{array}{l}
\forall\, h = \langle\langle \iota_1, op_1\rangle, \omega_1\rangle ... \langle\langle \iota_{\#h}, op_{\#h}\rangle, \omega_{\#h}\rangle \in \mathcal{L}_{S^C}\ :\ \mathcal{I}(h) \in Tr(\mathcal{L}_{S^A}) \Rightarrow \\
\forall\, t_0, ..., t_{\#h} \in St^C\ : \\
(t_0 \in \mathit{INIT}^C \wedge (\forall\, 1 \leq i \leq \#h\ :\ (\langle \iota_i, t_{i-1}\rangle, \langle t_i, \omega_i\rangle) \in {op_i}^{S^C})) \Rightarrow \\
\exists\, R_0, ..., R_{\#h}, T_0, ..., T_{\#h}\ : \\
\quad a)\ R_0 = \mathit{INIT}^A \wedge T_0 = \mathit{INIT}^C \\
\quad b)\ \forall\ 0 \leq i \leq \#h\ :\ t_i \in T_i \wedge (T_i, R_i) \in ABS \\
\quad c)\ \forall\, 0 \leq i \leq \#h\ :\ \varnothing \neq R_i \subseteq \mathit{final}_{S^A}(h \uparrow i) \\
\quad d)\ \forall\, 1 \leq i \leq \#h\, \forall\, s_1 \in R_i\, \exists\, s_0 \in R_{i-1}\ :\ (\langle \iota_i, s_0\rangle, \langle s_1, \omega_i\rangle) \in {op_i}^{S^A}
\end{array}
\tag{4}
$$

Base case ($h = \varepsilon$): This trivially holds because $\mathit{final}_{S^A}(\varepsilon) = \mathit{INIT}^A \neq \varnothing$ and $(\mathit{INIT}^C, \mathit{INIT}^A) \in ABS$.

Induction step ($h' = h\langle\langle \iota_{\#h'}, op_{\#h'}\rangle, \omega_{\#h'}\rangle \in \mathcal{L}_{S^C}$ with $\mathcal{I}(h') \in Tr(\mathcal{L}_{S^A})$): Let $t_0 \in {INIT}^C$, $t_1, ..., t_{\#h'} \in St^C$ with $(\langle \iota_i, t_{i-1}\rangle, \langle t_i, \omega_i\rangle) \in {op_i}^{S^C}$, $1 \leq i \leq \#h'$. By the induction hypothesis there are sets $R_0, ..., R_{\#h}$, $T_0, ..., T_{\#h}$ that satisfy the property (4). That is $\varnothing \neq R_{\#h} \subseteq final_{S^A}(h) \subseteq ptrace_{S^A}(\mathcal{I}(h))$, $(T_{\#h}, R_{\#h}) \in ABS$ and $\mathcal{I}(h)\langle \iota_{\#h'}, op_{\#h'}\rangle \in Tr(\mathcal{L}_{S^A})$. The specification S^A is demonic and therefore $R_{\#h} \subseteq ptrace_{S^A}(\mathcal{I}(h)) \subseteq pre_{S^A}(\langle \iota_{\#h'}, op_{\#h'}\rangle)$. With $(P2)$ we then can find $R' \in \mathbb{P}\, St^A$, $T' \in \mathbb{P}\, St^C$ such that $R' \neq \varnothing$, $(T', R') \in ABS$ and $t_{\#h'} \in T'$. Furthermore, for any $s_1 \in R'$ there is a $s_0 \in R_{\#h}$ with $(\langle \iota_{\#h'}, s_0\rangle, \langle s_1, \omega_{\#h'}\rangle) \in {op_{\#h'}}^{S^A}$. Since $R_{\#h} \subseteq final_{S^A}(h)$ we can conclude that $R' \subseteq final_{S^A}(h')$. The induction step is completed by setting $R_{\#h'} = R'$ and $T_{\#h'} = T'$.

Finally, note that property (3) implies $Tr(\mathcal{L}_{S^A}) \subseteq Tr(\mathcal{L}_{S^C})$ and from property (4) we can infer $\Omega_{\mathcal{L}_{S^C}}(\tau) \subseteq \Omega_{\mathcal{L}_{S^A}}(\tau)$ for every trace $\tau \in Tr(\mathcal{L}_{S^A})$. This implies $\mathcal{L}_{S^C} \Subset \mathcal{L}_{S^A}$. □

As a corollary, power-refinement $\sqsubseteq$ defines a preorder on the set of demonic specifications.

5 Conclusions and Related Work

In this paper, we have extended the results on the refinement of demonic languages from [10]. We have presented a lattice structure for the set of demonic languages and proved the correspondence to the set of demonic specifications. By lifting the common notion of abstraction relations on state spaces to relations on the power sets of state spaces, it was possible to derive a single complete refinement rule. This refinement rule provides a characterisation of the ordering on demonic languages in terms of a refinement preorder on demonic specifications. This complements the classical characterisation by backward and forward refinement proven in [10].

The refinement ordering we use is consistent with the conventional semantics underlying a system specification in Z and VDM [14,13]. Hence the forward and backward refinement techniques of these specification languages are sound and complete methods for the verification of the refinement ordering $\Subset$ on demonic languages. To our knowledge, there is no refinement theory founded on abstraction relations on power sets of states. All relational approaches we are aware of work with abstraction relations on state spaces and therefore need at least two refinement rules for completeness results [5,9,7,6,11,14,13] or significant restrictions on the languages and specifications. Furthermore, those approaches are founded on the subset ordering on prefix-closed languages.

The restriction to demonic specifications and the generalisation to abstraction relations on power sets make it possible to obtain a similar completeness result as in the predicate transformer setting [2] with only one refinement technique. As for predicate transformers, finding abstraction relations with a single complete rule is in general more difficult than relying on the combination of two less complex rules. Hence for practical applications the combination of backward and forward refinement rules is recommended. Comparing the predicate transformer approach with our approach, Rewitzky and Brink [12] show that

predicate transformers can be interpreted as total functions on the power sets of the state spaces. The predicate transformer refinement ordering can then be seen as the subset ordering on the underlying behaviours, which implies that traces can disappear during the refinement process. Any cosimulation may be seen as a total function on the power sets of the state spaces. Our refinement ordering on demonic languages and specifications differs from the subset ordering in that traces cannot disappear during the refinement. Furthermore, a demonic composition principle in specifications is more general than the composition by total operations. Also, in our relational approach, a restriction to abstraction functions on power sets would be insufficient.

Acknowledgements: We thank Ian Hayes and the anonymous referees for their suggestions on earlier versions of the paper.

References

1. R. J. R. Back. On correct refinement of programs. *Journal of Computer and System Sciences*, 23:49–68, 1981.
2. P. H. B. Gardiner and C. Morgan. A single complete rule for data refinement. *Formal Aspects of Computing*, 5(4):367–382, 1993.
3. C.A.R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, 1972.
4. C.A.R. Hoare. *Communicating sequential processes.* Prentice-Hall International, UK, LTD, 1985.
5. C.A.R. Hoare, He Jifeng, and J. W. Sanders. Prespecifications in data refinement. *Information Processing Letters*, 25:71–76, 1987.
6. He Jifeng. Process simulation and refinement. *Formal Aspects of Computing*, 1:229–241, 1989.
7. He Jifeng. Various simulations and refinements. In J.W. de Bakker, C., W.P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 340–360. Springer-Verlag, 1989.
8. C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, second edition, 1990.
9. M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.
10. K. Lermer and P. Strooper. Refinement and state machine abstraction. Technical Report 00-01, Software Verification Research Centre, January 2000. To appear in *Theoretical Computer Science.*
11. N. Lynch and F. Vaandrager. Forward and backward simulation for timing-based systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 397–445. Springer-Verlag, 1991.
12. I. Rewitzky and C. Brink. Predicate transformers as power operations. *Formal Aspects of Computing*, 7(2):169–182, 1995.
13. W.-P. Roever and K. Engelhardt. *Data refinement: model-oriented proof methods and their comparison.* Cambridge tracts in theoretical computer science; 4. Cambridge University Press, 1998.
14. J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall, 1996.

Reasoning about Composition Using Property Transformers and Their Conjugates

Michel Charpentier[1] and K. Mani Chandy[2]

[1] University of New Hampshire
Computer Science Department
charpov@cs.unh.edu
[2] California Institute of Technology
Computer Science Department
mani@cs.caltech.edu

Abstract. Compositional design is concerned with both constructing systems by composing components and with deconstructing systems into proposed sets of components. In bottom-up design, engineers prove system properties given properties of components and a compositional structure. In top-down design, they propose properties of components and a compositional structure given system properties. In this paper we show how the theory of predicate transformers, which has been used so successfully in sequential programming, can be applied to compositional design of systems. The rules of composition we study are more general than the rules employed in sequential programming, and the systems we study are not limited to programs. We exploit theorems about weakest and strongest solutions to equations to obtain a collection of useful predicate transformers, and then we exploit the theory of conjugate transformers to obtain more useful transformers. We show how these transformers are useful for both bottom-up and top-down design.

1 Motivation

1.1 Composition and Compositional Properties

Composition is the most fundamental operation in design. Designers of space vehicles, buildings and programs have common concerns: How to compose systems from components and how to partition systems into components. Compositional design offers the hope of managing complexity by avoiding unnecessary detail: systems designers prove properties of systems given properties, but not detailed implementations, of components.

We introduce an informal concept of *compositional properties* to motivate our exploration, and define terms precisely later. Compositional properties are those classes of properties that allow designers to deduce system properties from component properties using simple rules. For example, mass is a compositional property because the mass of a system can be deduced in a simple way from the masses of components: the system mass is the sum of component masses. By

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 580–595, 2000.
© Springer-Verlag Berlin Heidelberg 2000

contrast, temperature does not appear to be a compositional property because the temperature of a system depends in very complex ways on the shapes, masses, insulation properties, power consumption, locations of the components, etc.

Designers have to compute properties of composed systems given properties of components, whether the properties are compositional or not. The challenge is to develop theories that help designers prove system properties they need from component properties.

In this paper, we restrict ourselves to properties that are predicates on systems. We explore properties that are compositional in the following sense: we can prove that a property holds for a system given that the property holds for some or all of its components. In later papers we propose to explore other kinds of compositional properties.

In our paper, systems are abstract entities. They are not necessarily programs and they may not have "states" or "computations." We consider composition operators that have certain algebraic properties, such as associativity, and we explore theorems that are derived solely from these properties. Our goal is to explore composition in the abstract as opposed to studying how composition is used in constructing specific kinds of systems.

The simplest rules are those that establish that a property X holds for a system given that (i) property X holds for at least one component, or (ii) property X holds for all components. Therefore, in this paper, we restrict attention to two kinds of compositional properties: *existential properties* and *universal properties*. A property is an existential property exactly when, for all systems, a system has the property if there exists a component of the system that has the property. A property is a universal property exactly when, for all systems, a system has the property if all components of the system have the property.

1.2 An Introduction to Property Transformers for Composition

We motivate our exploration of predicate transformers by a few examples, and then develop the theory.

Questions about the Weakest Existential Transformer. Consider the following specification S for a component F: All systems that have F as a component must have a property X.

We postulate that any system is a component of itself. Since F is a component of F it follows that F itself must have property X. If X is an existential property, then from the definition of existential properties it follows that X holds for all systems that contain F as a component. Therefore, if X is an existential property, the given specification S is equivalent to the simpler specification: X holds in F. What if property X is not existential?

Suppose we can define a predicate transformer WE where WE.X is an existential property stronger than X. If we can demonstrate that component F has existential property WE.X, then any system that includes component F also has property WE.X, and therefore also enjoys the weaker property X. Let S' be

the following specification of F: WE.X holds in F. From the above argument, it follows that specification S' is stronger than specification S.

Examples of the kinds of questions that we wish to explore are the following. For a given property X, is there a *weakest* existential property at least as strong as X? And, if this weakest property exists and we define WE.X to be this property, then are specifications S' and S equivalent?

Questions about the Strongest Existential Transformer. Next, consider a dual set of questions. Given that system F has property X, what properties can we deduce about all systems that have F as a component? If X is existential, then all systems that have F as a component also have property X. But, what if X is not existential?

Suppose we can define a predicate transformer SE where SE.X is an existential property weaker than X. If F has property X then it also has the weaker property SE.X and, since SE.X is existential, all systems that contain F as a component also satisfy SE.X. The obvious question to explore is: Can we define SE.X to be the *strongest* existential property weaker than X? And if we can, is SE.X the strongest property that holds for all systems that contain F as a component?

Questions about the Conjugate Weakest Existential Transformer. Now, consider an engineer designing a system top down. The designer is given the specification that the system must have property X. The designer asks the question: Can I restrict myself to considering only those components that have a property Y? In other words, can we prove that any system that contains a component that does *not* have property Y also does *not* have property X? We will show that the conjugate of the weakest existential transformer is helpful in answering this question.

Questions about Universal Transformers. We also consider the analogous case for universal properties. We explore the following question: Is there a weakest property Y such that, if all components of any system have property Y, then the system has property X? If X is a universal property and all components have property X then the system itself has property X. So, since Y must be at least as strong as X, Y is the same as X. What if X is not universal?

We can introduce a predicate transformer WU with the requirement that WU.X is universal and stronger than X. If we can then prove that all components of a system have property WU.X then we can conclude that the system enjoys this property and hence also enjoys the weaker property X.

Can we require that WU.X be the *weakest* universal property stronger than X? We can show that we cannot do so because there does not exist, in general, a weakest universal property stronger than X. What are good ways of defining WU, then? We do not have answers to this question.

In the main body of this paper, we explore similar questions about strongest universal transformers and their conjugates.

2 Terminology and Notations

2.1 Composition

We postulate a composition operator denoted by $\circ$ and we restrict ourselves to systems built from a finite number of applications of that operator. We assume that $\circ$ is associative. We do not assume other properties such as symmetry or idempotency. We do not interpret systems, and we do not consider how systems are constructed by composing elemental or atomic systems. All that is relevant in this paper is that systems can be composed to obtain systems.

We postulate the existence of a binary relation $\surd$ and we only consider systems $F \circ G$ for those components F and G for which $F \surd G$ holds. We assume that $F \surd G$ denotes that F can be composed with G (in that order).

We assume that if the system $F \circ G \circ H$ can be constructed, then it can be constructed by first composing F with G and then composing the resulting system with H, or by first composing G with H and then composing the resulting system with F on the left. Specifically, we assume the following property of $\surd$, for any F, G and H:

$$F \surd G \wedge (F \circ G) \surd H \;\equiv\; G \surd H \wedge F \surd (G \circ H) \;.$$

We assume the existence of a *UNIT* component that satisfies the following axiom for all systems F:

$$UNIT \surd F \;\wedge\; F \surd UNIT \;\wedge\; (UNIT \circ F = F \circ UNIT = F) \;. \tag{1}$$

For some theorems, we need the additional axiom that the unit system cannot have non-unit components:

$$(F \circ G = UNIT) \;\equiv\; (F = UNIT) \wedge (G = UNIT) \;. \tag{2}$$

Most results presented in this paper do not require this additional axiom. However, some results do. These results are marked with a $^{\circledast}$ sign.

2.2 Membership Relation

We introduce a specific notation to denote that a system F is part of a system G:

$$F \lhd G \;\triangleq\; \langle \exists H, K : H \surd F \wedge H \circ F \surd K : G = H \circ F \circ K \rangle \;.$$

Note that, because of the axiom (1) on the *UNIT* element, $\lhd$ is a reflexive operator and $UNIT \lhd F$ is true for any F.

2.3 Properties and Specifications

Properties are point-wise predicates on systems. We treat a property as boolean-valued function with a single argument of type system. We use dot notation to denote function application, as in $f.x$ denotes the application of function f to x.

Therefore, for any property X and any system F, the notation $X.F$ denotes the boolean: system F has property X. Following [13], we use square brackets to denote that a predicate is "everywhere true". For a property X, $[X]$ is the boolean: property X holds in all systems.

We introduce two properties that are specific to the *UNIT* component, $UNIT_{=}$ ("to be the *UNIT*") and its negation $UNIT_{\neq}$ ("to be different from the *UNIT*"):

$$UNIT_{=} \;.\; F \;\triangleq\; (F = UNIT) \qquad \text{and} \qquad UNIT_{\neq} \;.\; F \;\triangleq\; (F \neq UNIT) \;.$$

2.4 Bags of Colored Balls

In order to illustrate the ideas presented in this paper, we use a simple model of components. In this model, systems are bags of colored balls. Composition corresponds to bag-union and the *UNIT* element is the empty bag. Note that this bag model satisfies axiom (2). Therefore, properties marked with $^{\circledast}$ can be used when reasoning on this model.

Bags can always be composed (i.e., the relation $\surd$ is always *true*) and composition is symmetric (Abelian monoid), but we do not rely on these additional properties of the model for they may not be true of more interesting models. For instance, sequential composition of programs is not symmetric and parallel composition of processes may not always be possible (for example, if a process references a local variable from another process).

3 Existential and Universal Properties

3.1 Existential Properties

A property X is *existential* (denoted by the boolean *exist.X*) if and only if X holds in all systems that contain a component that satisfies X:

$$exist.X \;\triangleq\; \langle \forall F, G : F \surd G : X.F \vee X.G \Rightarrow X \;.\; F \circ G \rangle \;.$$

The following results can be proved about existential properties:

$$\begin{gathered} exist.X \wedge exist.Y \;\Rightarrow\; exist.(X \wedge Y), \\ exist.X \wedge exist.Y \;\Rightarrow\; exist.(X \vee Y), \\ exist.X \Rightarrow (X \;.\; UNIT \equiv [X]), \\ exist.UNIT_{\neq}{}^{\circledast} \;. \end{gathered}$$

We define a function *guarantees*, from pairs of properties to properties. The property X *guarantees* Y holds for a system F if and only if, for all systems G that contain F as a component, if X holds for G then Y also holds for G:

$$X \textit{ guarantees } Y \;.\; F \;\triangleq\; \langle \forall G : F \lhd G : X.G \Rightarrow Y.G \rangle \;.$$

Proposition 1 *For all properties X and Y,*

$$exist.(X \ guarantees \ Y) \ .$$

Thus, *guarantees* provides us with a systematic way of building existential properties from other kinds of properties. The existential properties described with *guarantees* have been used successfully in compositional specifications and proofs of distributed systems [7].

3.2 Universal Properties

A property X is *universal* (denoted by the boolean $univ.X$) if and only if X is true in all systems built from components all of which satisfy X:

$$univ.X \ \triangleq \ \langle \forall F, G : F \surd G : X.F \wedge X.G \Rightarrow X \ . \ F \circ G \rangle \ .$$

From the definitions of *exist* and *univ*, any existential property is also universal:

$$exist.X \Rightarrow univ.X \ .$$

Moreover, the following properties can be proved:

$$\begin{gathered} univ.X \wedge univ.Y \ \Rightarrow \ univ.(X \wedge Y), \\ univ.X \wedge exist.Y \ \Rightarrow \ univ.(X \vee Y), \\ univ.UNIT_{=} \ . \end{gathered}$$

3.3 All-Components and Some-Component Properties

We define two additional forms of composition as the duals of existential and universal composition. A property is *all-components* if and only if its negation is existential:

$$all\text{-}c.X \ \triangleq \ exist.(\neg X) \ .$$

Unfolding the definition of *exist*, we find that a property is an all-components property if and only if, when it holds for any system, it holds for all components of that system:

$$all\text{-}c.X \ \equiv \ \langle \forall F, G : F \surd G : X \ . \ F \circ G \Rightarrow X.F \wedge F.G \rangle \ .$$

In the same way, we define *some-component* properties as the duals of universal properties:

$$some\text{-}c.X \ \triangleq \ univ.(\neg X) \ .$$

A property is a some-component property if and only if, when it holds for any system, it holds for at least one component of that system:

$$some\text{-}c.X \ \equiv \ \langle \forall F, G : F \surd G : X \ . \ F \circ G \Rightarrow X.F \vee F.G \rangle \ .$$

Existential and universal properties are used in bottom-up design. Their dual are used in top-down design.

- **Bottom-up.** Given components that have existential or universal properties, designers can deduce properties of systems composed from these components using very simple rules.
- **Top-down.** Given that a system should satisfy an all-components property, designers know that they must design all components to also satisfy that property. Likewise, given that a system should have a some-component property, designers know that they must design at least one component to have that property.

Of course, systems may be specified in terms of properties that are not existential, universal, all-components or some-component. However, as shown in the next sections, any property can be related in a systematic way to some properties that have one of these characteristics.

3.4 Examples of Properties

In the bags of colored balls model, the following properties are examples of existential, universal, all-components and some-component properties:

exist	.	(at least one blue ball in the bag)	(3)
exist	.	(at least two balls of different colors in the bag)	
univ	.	(no ball in the bag is blue)	(4)
univ	.	(all the balls in the bag are red, or at least two are blue)	
all-c	.	(all the balls in the bag are blue, or all the balls are red)	
all-c	.	(there are at most three balls in the bag)	
some-c	.	(the bag contains more blues balls than red balls)	(5)
some-c	.	(exactly one ball in the bag is red)	

Note that (3) is also some-component, (4) is also all-components and (5) is also universal. All other example properties only have the stated characteristic (besides the fact that any existential property is universal and any all-components property is some-component).

4 Property Transformers for Composition

4.1 Extreme Solutions of Equations in Predicates

As claimed in sect. 3, conjunctions of universal properties are universal and conjunctions and disjunctions of existential properties are existential[1]. An equation in predicates has a weakest solution if and only if the disjunction of all solutions is itself a solution, and in this case the weakest solution is that disjunction. Likewise, an equation in predicates has a strongest solution if and only if the conjunction of all solutions is itself a solution, and in this case the strongest solution is that conjunction [13].

[1] In sect. 3, junctivity is stated finitely for convenience, but actually holds for an infinite number of predicates.

4.2 The Property Transformer WE

We consider the following equation in Z, parametrized by predicate X:

$$Z : [Z \Rightarrow X] \wedge \mathit{exist}.Z \ . \tag{6}$$

A property Z is solution of (6) if and only if it is existential and stronger than X. Since disjunctions of existential properties are existential, equation (6) has a weakest solution, and we denote that weakest solution by WE.X:

$$\mathrm{WE}.X \ \triangleq \ \langle \exists Z : [Z \Rightarrow X] \wedge \mathit{exist}.Z : Z \rangle \ .$$

For any property X, WE.X is the weakest existential property stronger than X.

Suppose we wish to design a system F so that any system that has F as a component enjoys property X. What properties must system F have? The next theorem tells us that the necessary and sufficient specification of such a component F is that it satisfies WE.X.

Proposition 2 WE.X *is the weakest property of a component that ensures that any system that contains that component will enjoy property* X*:*

$$\langle \forall F, G : F \lhd G : Y.F \Rightarrow X.G \rangle \ \equiv \ [Y \Rightarrow \mathrm{WE}.X] \ .$$

This is a consequence of the following proposition that states that, for a component, to bring the property X to any system containing the component, or to satisfy WE.X are equivalent.

Proposition 3 $$\mathrm{WE}.X \ . \ F \ \equiv \ \langle \forall G : F \lhd G : X.G \rangle \ .$$

Proposition 3 is proved in [8]. The following elementary properties hold as well:

$$[\mathrm{WE}.X \Rightarrow X],$$
$$\mathit{exist}.X \ \equiv \ [\mathrm{WE}.X \equiv X],$$
$$[\mathrm{WE}.(X \wedge Y) \ \equiv \ \mathrm{WE}.X \wedge \mathrm{WE}.Y], \tag{7}$$
$$[X \Rightarrow Y] \Rightarrow [\mathrm{WE}.X \Rightarrow \mathrm{WE}.Y], \tag{8}$$
$$[\mathrm{WE}.X] \equiv [X] \ .$$

Property (7) states that the transformer WE is conjunctive. It can be shown that WE is *not* disjunctive. Property (8) expresses that WE is monotonic.

The property transformer WE can be used in two ways. Firstly, it provides us with an abstract way of expressing that a component, all by itself, ensures a system property X. Moreover, proposition 3 can be used to derive proof rules for properties of the form WE.X for a specific formalism. For instance, proof rules for UNITY logic are used in the correctness proof of a distributed system in [7]. Secondly, properties such as (7) and (8) can be used to reason about existential properties in general, without the inconvenience of a quantification over components. For instance, using proposition 3, it is easy to see that

$$[X \ \mathit{guarantees} \ Y \ \equiv \ \mathrm{WE}.(X \Rightarrow Y)] \ .$$

Then, it is easier to reason on WE to deduce properties of *guarantees* (such as existentiality or transitivity).

4.3 The Property Transformer SE

The property transformer WE was defined using the fact that disjunctions of existential properties are existential. Because conjunctions of existential properties also are existential, the equation $Z : [X \Rightarrow Z] \wedge \mathit{exist}.Z$ has a strongest solution $\mathrm{SE}.X$:

$$\mathrm{SE}.X \;\triangleq\; \langle \forall Z : [X \Rightarrow Z] \wedge \mathit{exist}.Z : Z \rangle \;\; .$$

For any property X, $\mathrm{SE}.X$ is the strongest existential property weaker than X. Among interesting properties concerning SE, we can prove:

$$\begin{gathered}
[X \Rightarrow \mathrm{SE}.X], \\
\mathit{exist}.X \;\equiv\; [\mathrm{SE}.X \equiv X], \\
[\mathrm{SE}.(X \vee Y) \;\equiv\; \mathrm{SE}.X \vee \mathrm{SE}.Y], \\
[X \Rightarrow Y] \Rightarrow [\mathrm{SE}.X \Rightarrow \mathrm{SE}.Y], \\
[\mathrm{SE}.X] \equiv (X \;.\; \mathit{UNIT})^{\circledast} \;\; .
\end{gathered} \tag{9}$$

Property (9) has an interesting intuitive explanation. $\mathrm{SE}.X$ holds for all systems that have at least one component that satisfies X. Since all systems have *UNIT* as a component, for all properties X that hold for *UNIT*, all systems have property $\mathrm{SE}.X$. So, as expected, designers do not get useful information from knowing that *UNIT* is a component of their systems.

Proposition 4 *$\mathrm{SE}.X$ is the strongest property that can be deduced of any system built from components, one of which, at least, satisfies X:*

$$\langle \forall F, G : F \lhd G : X.F \Rightarrow Y.G \rangle \;\equiv\; [\mathrm{SE}.X \Rightarrow Y] \;\; .$$

4.4 The Property Transformer SU

From sect. 3, we know that conjunctions of universal properties are universal. However, disjunctions of universal properties are not always universal. Therefore, we cannot define a transformer WU in the same way as we defined WE. Actually, it can be shown [8] that some properties do not have a weakest universal property stronger than them. Nevertheless, because conjunctions of universal properties are universal, the equation $Z : [X \Rightarrow Z] \wedge \mathit{univ}.Z$ has a strongest solution $\mathrm{SU}.X$:

$$\mathrm{SU}.X \;\triangleq\; \langle \forall Z : [X \Rightarrow Z] \wedge \mathit{univ}.Z : Z \rangle \;\; .$$

For any property X, $\mathrm{SU}.X$ is the strongest universal property weaker than X. Because $\mathrm{SE}.X$ is universal (since it is existential) and weaker than X, we can deduce:

$$[\mathrm{SU}.X \Rightarrow \mathrm{SE}.X] \;\; .$$

Among interesting properties concerning SU, we can prove:

$$\begin{gathered}[X \Rightarrow \mathrm{SU}.X], \\ \mathit{univ}.X \;\equiv\; [\mathrm{SU}.X \equiv X], \\ [X \Rightarrow Y] \Rightarrow [\mathrm{SU}.X \Rightarrow \mathrm{SU}.Y], \\ [\mathrm{SU}.X] \Rightarrow (X \;.\; \mathit{UNIT})^{\circledast} \;. \end{gathered} \tag{10}$$

Note that property (10) is only an implication because SU.X is, in general, strictly stronger than SE.X. Also, SU is monotonic but neither conjunctive, nor disjunctive.

If all components of a system satisfy X, they also satisfy SU.X and, because SU.X is universal, the whole system satisfies SU.X. Moreover, SU.X is the strongest property that can be deduced this way.

Proposition 5 SU.X *is the strongest property that can be deduced of any system built from components that all satisfy* X*:*

$$\begin{aligned} \langle \forall F, G, H, \cdots : F \surd G \wedge F \circ G \surd H \wedge F \circ G \circ H \surd \cdots : \\ X.F \wedge X.G \wedge X.H \wedge \cdots \Rightarrow Y \;.\; F \circ G \circ H \circ \cdots \rangle \\ \equiv [\mathrm{SU}.X \Rightarrow Y] \;. \end{aligned}$$

4.5 Example

We consider the following question: What must be proved on a bag of balls to ensure that any system that contains that bag, if it contains at least one red ball, contains at least two balls of different colors? Formally, the specification of the component is that any system that contains that component satisfies:

$$(\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors}) \;.$$

Then, from proposition 2, we know that the specification for the component is:

$$\mathrm{WE}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \;. \tag{11}$$

In this section, we show how an explicit formulation of property (11) can be calculated. The calculation relies on the following proposition, proved in [8]:

Proposition 6 $\qquad [X \equiv \mathit{UNIT}_{\neq}] \vee [\mathrm{WE}.(\mathit{UNIT}_{=} \vee X) \equiv \mathrm{WE}.X]^{\circledast} \;.$

Then, we can calculate an equivalent formulation of (11):

$$\begin{aligned} & \mathrm{WE}.((\text{at least 1 red ball}) \Rightarrow (\text{at least 2 colors})) \\ = \;& \{\text{Predicate calculus}\} \\ & \mathrm{WE}.(\mathit{UNIT}_{=} \vee (\text{at least 1 non red ball})) \\ = \;& \left\{ \begin{array}{l} \text{Assume there exist red balls, hence} \\ \neg[(\text{at least 1 non red ball}) \equiv \mathit{UNIT}_{\neq}], \text{ apply prop. 6} \end{array} \right\} \end{aligned}$$

$$
\begin{array}{l}
\quad \mathrm{WE}.(\text{at least 1 non red ball}) \\
= \{\mathit{exist}.(\text{at least 1 non red ball})\} \\
\quad (\text{at least 1 non red ball}) \quad .
\end{array}
$$

Therefore, we deduce that the necessary and sufficient specification of the component is that it should contain at least one non-red ball. In other words, to ensure that any system that uses F as a component will have at least two balls of different colors provided it has at least one red ball, it is sufficient and necessary that F contains at least one non-red ball. This is consistent with our intuition. However, instead of guessing the desired property of F and then prove that it is both necessary and sufficient, we have *calculated* the property. This provides us with both the property and the proof at the same time, and avoids dealing explicitly with the universal quantification over components.

5 Conjugates of Property Transformers

Any predicate transformers has a unique *conjugate*. Duality allows us to easily deduce properties of a predicate transformer from properties of its conjugate. In this section, we focus on the conjugates of WE, SE and SU.

5.1 Conjugate of a Predicate Transformer

Let f be a predicate transformer. Its conjugate, denoted by f^* is defined by [13]:

$$f^*.X \;\triangleq\; \neg f.(\neg X) \quad .$$

Duality provides us with a way of deducing properties of f^* from properties of f, and vice-versa. For instance, f is monotonic iff f^* is monotonic. In the same way, f is conjunctive (resp. disjunctive) iff f^* is disjunctive (resp. conjunctive).

5.2 The Property Transformer WE*

We define WE^* as the conjugate of the property transformer WE:

$$\mathrm{WE}^*.X \;\triangleq\; \neg \mathrm{WE}.(\neg X) \quad .$$

Because the dual of existential properties are all-components properties, we can easily deduce that $\mathrm{WE}^*.X$ is the strongest all-components property weaker then property X. Moreover, applying the duality principle to proposition 2, we can deduce the following proposition.

Proposition 7 *$\mathrm{WE}^*.X$ is the strongest property that can be deduced of all components of any system that satisfies X:*

$$\langle \forall F, G : F \lhd G : X.G \Rightarrow Y.F \rangle \;\equiv\; [\mathrm{WE}^*.X \Rightarrow Y] \quad .$$

Duality can also be applied to the basic properties of WE to obtain the following properties of WE^*:

$$
\begin{gathered}
[X \Rightarrow \mathrm{WE}^*.X], \\
\textit{all-c}.X \;\equiv\; [\mathrm{WE}^*.X \equiv X], \\
[\mathrm{WE}^*.(X \vee Y) \;\equiv\; \mathrm{WE}^*.X \vee \mathrm{WE}^*.Y], \\
[X \Rightarrow Y] \Rightarrow [\mathrm{WE}^*.X \Rightarrow \mathrm{WE}^*.Y], \\
[X \equiv \mathit{false}] \equiv [\mathrm{WE}^*.X \equiv \mathit{false}] \;\; .
\end{gathered}
$$

5.3 The Property Transformer SE*

In the same way as what is done in the previous section, we can study the conjugate of the property transformer SE. Applying duality, $\mathrm{SE}^*.X$ is the weakest all-components property stronger than X. From proposition 4, we obtain:

Proposition 8 *$\mathrm{SE}^*.X$ is the weakest property that must be proved on a system to ensure that all components of the system satisfy X:*

$$\langle \forall F, G : F \lhd G : Y.G \Rightarrow X.F \rangle \;\equiv\; [Y \Rightarrow \mathrm{SE}^*.X] \;\; .$$

5.4 The Property Transformer SU*

$\mathrm{SU}^*.X$, as the conjugate of $\mathrm{SU}.X$, is the weakest some-component property stronger than X. By duality of proposition 5:

Proposition 9 *$\mathrm{SU}^*.X$ is the weakest property that must be proved on a system to ensure that at least one component of the system satisfies X:*

$$
\begin{aligned}
\langle \forall F, G, H, \cdots : F \surd G \wedge F \circ G \surd H \wedge F \circ G \circ H \surd \cdots : \\
Y \;.\; F \circ G \circ H \circ \cdots \Rightarrow X.F \vee X.G \vee X.H \vee \cdots \rangle \\
\equiv [Y \Rightarrow \mathrm{SU}^*.X] \;\; .
\end{aligned}
$$

5.5 Comparison of the Six Transformers

The following table summarizes the intuitive interpretation for each one of the six property transformers that we have defined. Each transformer corresponds to a different view of composition:

WE.X	What must be proved on a component to ensure that any system that contains that component satisfies X.
SE.X	What can be deduced of any system that contains at least one component that satisfies X.
SU.X	What can be deduced of any system that contains only components that satisfy X.
WE*.X	What can be deduced on all components of any system that satisfies X.
SE*.X	What must be proved on a system to ensure that all components satisfy X.
SU*.X	What must be proved on a system to ensure that at least one component satisfies X.

5.6 Example

We consider another example that uses the bags of balls model: If a system contains exactly one ball and that ball is blue, what can we tell from its components? Using proposition 7, we can claim that all components in such a system must satisfy:

$$\text{WE}^*.(\text{exactly one ball and the ball is blue}) \ . \tag{12}$$

Applying duality to proposition 6, we deduce:

$$[X \equiv \mathit{UNIT}_{=}] \vee [\text{WE}^*.(\mathit{UNIT}_{\neq} \wedge X) \equiv \text{WE}^*.X]^{\circledast} \ .$$

We can then calculate and equivalent formulation for (12):

$$\begin{array}{ll} & \text{WE}^*.(\text{exactly one ball and the ball is blue}) \\ = & \{\text{Predicate calculus}\} \\ & \text{WE}^*.(\mathit{UNIT}_{\neq} \wedge (\text{all balls are blue}) \wedge (\text{at most one ball})) \\ = & \{\text{Apply the dual of prop. 6}\} \\ & \text{WE}^*.((\text{all balls are blue}) \wedge (\text{at most one ball})) \\ = & \left\{\begin{array}{l}\text{Both properties are all-components and a conjunction of all-components} \\ \text{properties is an all-components property, for which } [\text{WE}^*.X \equiv X]\end{array}\right\} \\ & (\text{all balls are blue}) \wedge (\text{at most one ball}) \ . \end{array}$$

This is consistent with our intuition: If a system contains only one ball and the ball is blue, then all its components must contain only blue balls and at most one ball. Note that the condition we obtain is no guaranteed to hold in a system even if it holds in all its components because the property "at most one ball" is not universal.

6 Conclusions

This paper reports on an ongoing exploration of using ideas from the mathematics of program construction to develop a theory of compositional design. This paper shows how we exploit concepts from the axiomatic semantics of programs for designing systems. The specific constructs that we investigated in this paper are predicate transformers and their conjugates. The only assumptions we made about the composition operator were that it was associative and that it had a unit element. A sizable body of theory about transformers can be developed from only these limited assumptions.

We started this study because of our conviction of the importance of composition in systems design. We believe that systems, and especially software systems, should more and more be constructed from generic, "off the shelf", components. This means that *reuse* of systems and components is going to be a central issue.

Reuse of existing systems and description of generic components require a specification language that is abstract enough to be able to specify only the relevant aspects of a system and to hide operational details as much as possible. For this reason, we depart from the process calculus approach, such as in CSP or CCS, and focus on logical specifications. Especially, we are interested in applying our approach to concurrent systems specified with temporal logics.

Of course, more abstract specifications lead to more difficulties in terms of composition. However, a great amount of work has been done in relation with the composition of concurrent systems described in terms of temporal logic specifications. Among the logics that were considered, we can cite the linear temporal logic [15] and some variants [14], TLA [2] or UNITY [9,10,11,12,16]. It should be noted that all these works, and even more general studies done at a semantic [1, 4] or syntactic [3] level, rely on the same fundamental hypothesis that systems are described in terms of states and computations. More precisely, the key point that allows systems to be composed is always the same: Components are specified in terms of *open computations*, i.e., infinite traces that are shared with the environment.

In this paper, we adopt a dual perspective on the question of composition. We do not want to consider systems that can always be composed (i.e., that are specified in such a way that composition is going to work) and we do not rely on the open computations assumption. Instead, we consider *any* kind of systems and look at what happens when they are composed. In this way, we hope to be able to reason on composition in the abstract and understand its fundamental issues.

So far, this proposed framework helped us better understand the *guarantees* operator, which was originally defined in [5], along with existential and universal properties. It is interesting to note that *guarantees*, when applied to temporal logic and reactive systems, gives us a powerful way to combine logical specifications. Especially, the compositional characteristics (existential) of X *guarantees* Y does not depend on properties X and Y. For instance, both the left-hand side and the right-hand side can be progress properties, which is not

possible with usual assumption-commitment specifications. This leads to simpler proofs of composition, as in [7].

Also, the framework emphasizes a symmetry between top-down design and bottom-up design. Conjugates of predicate transformers allow us to deduce theorems about top-down decomposition from corresponding theorems of bottom-up composition. We believe that, from a practical point of view, the top-down problem (identify suitable components) is as important as the more classical bottom-up problem (deduce system properties).

In this paper we limited our discussion to properties that were existential or universal or conjunctions of such properties. We have also shown elsewhere [6,7] that nice proofs of correctness for significant concurrent programs can be developed using these concepts coupled with a logic such as UNITY. In this paper we were not concerned with showing how these concepts could be used for programming; instead, we were primarily concerned with showing how concepts from programming can be applied to broad classes of systems in which composition has simple properties such as associativity and existence of a unit element.

Much further work needs to be done to develop an axiomatic semantics of a theory of compositional design. We have only begun to explore the area. Theorems that derive from further assumptions about the compositional operator must be developed. Properties that are not necessarily predicates on systems should be studied. In general, a property is a function from systems to some type that is not limited to booleans. A theory should be able to reason about properties such as mass and energy consumed. We believe that it is possible to construct axiomatic theories that help in understanding the basic principles of compositional design.

References

1. Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, January 1993.
2. Martín Abadi and Leslie Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, May 1995.
3. Martín Abadi and Stephan Merz. An abstract account of composition. In Jivrí Wiedermann and Petr Hajek, editors, *Mathematical Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, pages 499–508. Springer-Verlag, September 1995.
4. Martín Abadi and Gordon Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, June 1993.
5. K. Mani Chandy and Beverly Sanders. Reasoning about program composition. Submitted for publication.
   http://www.cise.ufl.edu/~sanders/pubs/composition.ps.
6. Michel Charpentier and K. Mani Chandy. Examples of program composition illustrating the use of universal properties. In J. Rolim, editor, *International workshop on Formal Methods for Parallel Programming: Theory and Applications (FMPPTA'99)*, volume 1586 of *Lecture Notes in Computer Science*, pages 1215–1227. Springer-Verlag, April 1999.

7. Michel Charpentier and K. Mani Chandy. Towards a compositional approach to the design and verification of distributed systems. In J. Wing, J. Woodcock, and J. Davies, editors, *World Congress on Formal Methods in the Development of Computing Systems (FM'99), (Vol. I)*, volume 1708 of *Lecture Notes in Computer Science*, pages 570–589. Springer-Verlag, September 1999.
8. Michel Charpentier and K. Mani Chandy. Theorems about composition. Technical Report CS-TR-99-02, California Institute of Technology, January 2000. 29 pages.
9. Pierre Collette. *Design of Compositional Proof Systems Based on Assumption-Commitment Specifications. Application to* UNITY. Doctoral thesis, Faculté des Sciences Appliquées, Université Catholique de Louvain, June 1994.
10. Pierre Collette. An explanatory presentation of composition rules for assumption-commitment specifications. *Information Processing Letters*, 50:31–35, 1994.
11. Pierre Collette and Edgar Knapp. Logical foundations for compositional verification and development of concurrent programs in UNITY. In *International Conference on Algebraic Methodology and Software Technology*, volume 936 of *Lecture Notes in Computer Science*, pages 353–367. Springer-Verlag, 1995.
12. Pierre Collette and Edgar Knapp. A foundation for modular reasoning about safety and progress properties of state-based concurrent programs. *Theoretical Computer Science*, 183:253–279, 1997.
13. Edsger W. Dijkstra and Carel S. Scholten. *Predicate calculus and program semantics*. Texts and monographs in computer science. Springer-Verlag, 1990.
14. J.L. Fiadeiro and T. Maibaum. Verifying for reuse: foundations of object-oriented system verification. In I. Makie C. Hankin and R. Nagarajan, editors, *Theory and Formal Methods*, pages 235–257. World Scientific Publishing Company, 1995.
15. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
16. Rob T. Udink. *Program Refinement in* UNITY*-like Environments*. PhD thesis, Utrecht University, September 1995.

Some New Directions in the Syntax and Semantics of Formal Languages

Gordon D. Plotkin

Department of Computer Science
University of Edinburgh
Edinburgh, EH9 3JZ, SCOTLAND
gdp@dcs.ed.ac.uk

(We regret that the paper was not available at the time of publication of the proceedings.)

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, p. 596, 2000.
© Springer-Verlag Berlin Heidelberg 2000

New Challenges for Theoretical Computer Science
(General Introduction to the Panel)

Jozef Gruska*

Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
gruska@www.fi.muni.cz

Abstract. This is a general introduction to the panel *New Challenges for Theoretical Computer Science* with panelists G. Ausiello, J. Gruska, U. Montanari, T. Nishizeki, Y. Toyama and J. Wiedermann. The aim of the panel is to point out and analyse new challenges theoretical computer science has to deal with in order to develop properly as fundamental science and in order to respond well to needs coming from the outside world, especially to those informatization of society and global networking bring.

Vision Theoretical computer science still keeps being developed too much according to the same principles as pure mathematics – following mainly the inherent needs to develop more powerful concepts and proof methods, to generalize results, to simplify proofs, and to solve important/interesting or long standing problems. Most of the pure mathematicians hardly look to the outside world for its needs and motivations.

Theoretical computer science is believed here to have some needs to follow the same research paradigms as mathematics does, but to have ultimately also much broader goals: To study fundamental laws and limitations of the information processing world and also to contribute to the development of the new, the third, main methodology[1] of mankind in general and of science in particular. Moreover, theoretical computer science has a need and obligation to make its contribution to solving new big and difficult problems informatization and global communication bring. Theoretical computer science therefore needs more often and more carefully to look out of its theoretical vacuum to see whether it is concentrating correctly and enough on its main global aims, to see where the problems are it can and should help to solve and how really powerful are tools it develops.

* Support of the GAČR grant 201/98/0369 and of the grant CEZ:J07/98:143300001 is to be acknowledged.

[1] This refers to the understanding that so far there were two main methodologies science had: the theoretical one (much represented and supported by mathematics) and the experimental one. Computer Science is believed to bring a third methodology that at the same time reinforces (substantially) old methodologies and helps to bridge them.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 599–601, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Challenges At the break of the millennia theoretical computer science is faced with several new, scientifically fundamental and practically very important and hard challenges.

- Due to several developments one can witness a liberalization of theoretical computer science from serving mainly computation and communication technology. Theoretical computer science becomes more and more clearly fundamental science that has to study the laws and limitations of the information processing world in all its forms and it has to concentrate on developing corresponding foundations, paradigms, concepts, models and tools.
- An understanding of the limitations of current computing and communication technologies and methods has turned attention to a more systematic study of the ways Nature can perform information processing. The goals are two sided: To develop new, more Nature based, information processing tools and also to get a better understanding of Nature. Quantum computing, molecular computing and brain computing are among areas that keep bringing new challenges fascinating in both directions.
- Radically new questions concerning computing and its power starts to be seriously asked and radically new theories and computational models need and start to be on the agenda. They range from theories and models connected with new applications and new computational modes to theories and computational models that try to understand brain, to be more powerful than "Turing models", to capture the essence of learning and understanding and even to capture the essence of evolution and genesis of Universe. And not only that. Similarly as in physics the question "Is there a (computational) model of everything" is being raised.
- Since society is increasingly dependent, see [2], on very large software systems and on global communications, reliability of very big software systems and security of communications are becoming another very hard and very important problems theoretical computer science is asked and obliged to help to deal with – by developing corresponding insights, concepts, theories and methods.
- The existence of global networks and their size and complexity require to develop new theoretical tools to handle new issues created by them. For example, the task is to develop abstract semantics framework to deal with such problems of highly distributed networks as those created by needs to manage cooperation, control and synchronization as well as verification of systems with high mobility of codes (created in different programming languages) and computations.
- Various limitations concerning information processing have been discovered. Some of them are very unpleasant. For example enormous number of practically important problems are NP-complete. An important task is to find ways how to compute, in some reasonable sense, what cannot be computed in some very strict sense and worst case.
- A variety of new challenges for theoretical computer science come also with fast growing needs to solve (efficiently) problems with enormously large data.

New dimensions to this problem are due to the facts that these data may be produced in a very distributed way and can keep being dynamically changed. In addition, they need to be solved in an on-line way. This in turn brings to a new dimension problems of efficient distributed computing and of efficient use of hierarchies of (secondary) memories. Searching for textual, visual and sound data in wide area networks, with millions of nodes and all-to-all connections, as well as broadcasting of such data in global networks are examples of such problems.

Some of these challenges are discussed in more details in position statements of panelists.

Teaching of theoretical computer science seems to be in a crises. Students keep challenging relevance of some subjects and ways theory is presented. Theoretical computer science seems to have needs to learn from other mature sciences, as physics, chemistry and biology to present its concepts, methods and results in much more synthetizing form and to concentrate mainly on outcomes with clear importance (see [1], for an attempt to do that). New challenges theoretical computer science is to deal with require also that its fundamental and powerful of concepts, methods and results are presented to students in a much more mature and impressive way.

Epilogue Theoretical computer science research should get out of its "concentration on $\lg^* n$ improvements" and its education should be such that students do not remember it mainly as a "pumping-lemmas stuff". Big new challenges of theoretical computer science and enormous potentials it has require that.

References

1. Jozef Gruska. *Foundations of computing.* Thomson International Computer Press, 1997.
2. President's Information Technology Advisory Committee. Information. Informaation technology research: Investing in our future. Technical report, Report to the President, National Coordination Office for Computing, Information and Communication, 1999.

Algorithm Design Challenges
(Position Statement)

Giorgio Ausiello

Universita "La sapienza" Roma
Via Salaria 113, I-00198 Roma, Italy
ausiello@dis.uniroma1.it

Considering the tremendous growth of computer science in the last twenty five years, since the first personal computer made its appearance, and the even stronger impetus that to this growth was given by the creation of the World Wide Web, I do not think that anybody could dare to say what computing will be like during the 21st century. We are by now used to quite unexpected technological leaps that change the picture that is in our hands while we are still trying to analyze it and we do not know what else is waiting us behind the corner. May be, although I am quite skeptical, that some technological breakthrough will make molecular computers or quantum computers available earlier than we might expect and that this might dramatically change our view of computing.

On the other side, on a more conservative ground, if we look at a closer time scale, the next ten years for example, some driving trends of computing appear rather clearly. We may then be able to foresee the evolution of computing on the basis of the current directions of technology and applications and, accordingly, to pinpoint some of the contributions that theoretical research can give to such evolution.

First of all, the most impressive push to the technological evolution of computer science derives from the "all connected to all" paradigm and the use of IP as the basic support for this paradigm. This enormously increases the request for guaranteed quality of service performances on the internet in terms of voice and image communication, broadcasting, searching and web caching, correctness of mobile code migrations, security etc. Among other problems this development implies the study and design of new efficient and powerful techniques for search engines as it is witnessed by the Search Engine Watch and by the fact that important research institutions have started projects in this area such as the Clever Project (IBM, San Jose') and the Google Project (Stanford University).

An interesting aspect of the scientific development in this field is related to the fact that, in order to design web searching engines we need to represent and manipulate large portions of the web, a graph with millions of node and billions of arcs. This leads us to the second characteristic aspect of the current evolution of Information Technology: the need to manipulate very large input data. While software applications for the final user become (apparently at least) simpler and simpler, the applications involved with the management of ITC systems become more and more complex and resource demanding. For example, for defining their billing strategies telecommunication companies want to know the communication graph of all calls among their customers. Also scientific computer applications

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 602–603, 2000.
© Springer-Verlag Berlin Heidelberg 2000

require to deal with huge data sets (very long strings in biology, very large matrices in theoretical physics, very large instances for finite element methods in fluid dynamics etc). In all these cases the classical asymptotic analysis is not adequate for describing and interpreting the behaviour of programs. We have to rethink our computation models for allowing the analysis and design of algorithms on secondary storage and for predicting the impact of locality and cashing strategies on different algorithms. Also in this area, research efforts are already being carried out in various Universities worldwide, among others, Duke University in the US and the Universities participating in the European Union research project ALCOM-FT.

A third direction of computer evolution that can be clearly identified and that requires new answers from theoreticians is related to the increasing role of dynamic and on line applications. In this type of applications (think of routing problems, scheduling, resource allocation, investment strategies etc.) data are not static but evolve in a somewhat unpredictable way and we have to face such evolution by maintaining our data in such a way that we do not have to recompute from scratch the solution of a problem at any change of input data but we can just rearrange it with a limited cost. At the same time we want to define strategies for problem solution that, while not being optimal, anyhow are "competitive", that is do not lose too much with respect to an off line algorithm that, ahead of time, is aware of the whole sequence of changes in the input data. The efficient and competitive solution of dynamic and on line problems has been studied over the last fifteen years but still practical applications of theoretical results are limited by the fact that theoretical models are inadequate and a lot more research efforts should be spent in this domain.

Quantumization of Theoretical Informatics
(Position Statement)

Jozef Gruska*

Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
gruska@fi.muni.cz

Maturing of informatics Recent developments in the area of quantum information processing and communications (QICP) made clear that foundations of computing have to be based not on the laws and limitations of the classical physics as so far, but on the laws and limitations of quantum physics.

This discovery has several profound impacts on theoretical informatics, on its goals and methods and brings also a new series of the very basic challenges for theoretical informatics. This discovery has also significantly increased *liberalization of theoretical informatics* from being mainly a servant of computing and communication technology and applications and put it into the position of the fundamental science, that has also strong impact on creation and development of new computing and communication technologies. This liberalization is expected to have far reaching consequences on how we should and will see foundations, aims, paradigms and methods of theoretical informatics.

Another implication of the quantumization of (theoretical) informatics is that it helped to see more clearly *basic scientific goals of theoretical informatics*: to discover and study the laws and limitations of the information processing world (whatever it means). This can also be viewed as a parallel to the main goal of physics – to discover and to study the laws and limitations of the physical world (whatever it means).

Going quantum Theoretical informatics has already started to get involved in the study of problems of QIPC. Moreover, its contributions, especially concerning the power of quantum algorithms and communication protocols, security of cryptographic protocols, quantum error correction codes and fault tolerant computations, as well as the existence of efficient universal quantum Turing machines, have been crucial for the development of the field. However, one has also to see that theoretical informatics has concentrated so far mainly on problems that can be seen as natural quantum variations of the classical problems. For example, to study such models of quantum automata that can be seen as direct quantumizations of the classical models. In other words, theoretical informatics is going quantum, but only very slowly and cautiously is getting out of its old views of the information processing world and its principles, laws and limitations.

* Support of the GAČR grant 201/98/0369 and of the grant CEZ:J07/98:143300001 is to be acknowledged.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 604–608, 2000.
© Springer-Verlag Berlin Heidelberg 2000

The aim of this position statement is to point out that theoretically equally interesting, and perhaps even deeper and more important, are problems of quantum information processing of a different type, those that are inherently quantum, and to try to turn attention of (theoretical) informatics community to such problems for two reasons: (a) they are of large (key) importance for the development of the field; (b) the research paradigms and methods of (theoretical) informatics seem to be well suited to deal with these problems; (c) they are intellectually exciting and rewarding as well as fundamentally deep.

A related underlying thesis is that paradigms, concepts, methods and results of theoretical informatics are so fundamental that they should be expected to have a very broad range of applications, similarly as those of quantum mechanics.

Main new features theoretical informatics has to deal with when *going quantum* can be summarized as follows.

- New information processing and communication resources (quantum superposition, parallelism, channels, entanglement and measurement).
- New information processing primitives (qubits, unitary operators, projection measurements, density matrices, superoperators and POV measurements for computation and bits, qubits and ebits for communication.
- New concepts of feasibility and new computational complexity classes.
- New types of (quantum) gates, circuits, automata, algorithms and protocols.
- New concepts of (quantum) information and new (quantum) information theory.
- New concepts of security (based on the laws of physics).
- New goals: (a) Those oriented to quantum physics (for example, a deeper understanding of such quantum phenomena as measurement, non-locality, decoherence). (b) Those oriented to theoretical informatics (for example, development of complexity theory on the basis of the quantum information processing laws and limitations). (c) Those oriented to quantum information processing (for example, an understanding of the computational and communication power of entanglement, a development of quantum information theory and of new computation and communication paradigms).

Interesting and important enough, it has been to a large extend due to the insights and results on the computational power of quantum information processing that an understanding has emerged that enormous experimental and developmental effort clearly needed to build powerful quantum information processing systems could pay off. This was followed by an observation that the research in quantum information processing can contribute also to a new and deeper understanding of quantum physics and by that also of Nature. A contribution to the development of such an understandings can be seen as one of the major impacts of concepts, methods and results of theoretical informatics in general, and of (computational) complexity theory in particular, on other sciences.

Quantum information processing is an area where very fundamental questions concerning Nature are being (re)asked and need to be answered. An area

where one finds various mysterious and paradoxical phenomena. An area where it is very nontrivial to find mathematical equivalents to the basic concepts and phenomena of Nature and proper interpretations, in terms of Nature elements and phenomena, of the mathematical concepts being used.

Quantum resources Power of quantum information processing and communication is based on an appropriate use of several new quantum resources.

- *Quantum superposition.* An n qubit quantum register can be simultaneously in a superposition of all 2^n (standard) basis states $\{|x\rangle \mid x \in \{0,1\}^n\}$ of $\mathcal{H}_{2^n}$. In addition, an equally weighted superposition of all the basis states,

$$|\phi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle |0^{(n)}\rangle, \tag{1}$$

  an important initial state of many computations, can be created in a single step using only n simple quantum gates.
- *Quantum parallelism.* In one step of a quantum evolution exponentially many classical computations can be "performed". For example, for any function $f : \{0, 1, \ldots, 2^n - 1\} \to \{0, 1, \ldots, 2^n - 1\}$, there is a unitary mapping $U_f : |x, 0\rangle \to |x, f(x)\rangle$ and if U_f is applied to the exponentially large superposition $|\phi\rangle|0^{(n)}\rangle$ (see 1) of two n-qubit registers, then in one step the state

$$|\phi_f\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, f(i)\rangle \tag{2}$$

  is created and therefore exponentially many values of f can be "computed" in one step.
- *Quantum entanglement.* Some quantum states of a bipartite quantum systems cannot be expressed as tensor products of the states of its subsystems, for example the state $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$. Such pure states are called *entangled* and they are a puzzling phenomenon because they exhibit non-locality features. Indeed, it may be the case that two particles in an entangled state are far from each other. In spite of that any measurement of one of the particles "automatically" determines the state of the second particle.

Quantum limitations Quantum physics imposes also several severe limitations on quantum information processing.

- *Heisenberg's uncertainty principle* says that measuring the value of one observable more accurately makes the value of another, noncommutative, observable less certain. In addition, there is a certain intrinsic uncertainty with which values of two observables can be measured, and once a way of measurement is fixed, this uncertainty, in general, increases.

- *No cloning theorem.* Very important for quantum information processing is the result that an unknown quantum state cannot be cloned. (Namely, there is no unitary transformation U, such that for any one-qubit state $|\psi\rangle$, $U(|\psi, 0\rangle) = |\psi, \psi\rangle$.) No cloning theorem seems to be a bad news. For example, it was to a large extent due to this theorem that quantum error-correcting codes seemed to be impossible. However, no cloning theorem is also a good news. Due to the impossibility to copy quantum information safely we have quantum key generation systems that are provably unconditionally secure.
- *Holevo theorem.* It seems that a state of a quantum register can contain unlimited amount of information because amplitudes can be numbers with infinite decimal expansion. However, because of the peculiarities of quantum measurement the amount of information we can get out of a quantum state to the classical world is very limited. Holevo theorem says that into an n-qubit register state we can encode and later decode safely only n bits.
- *Decoherence.* This is a term for the processes of coupling of quantum systems (processor) with their environment, if they are not perfectly isolated. As a consequence, quantum states of the system are modified due to the interactions of the system with the environment. Such interactions mean that quantum dynamics of the environment is also relevant to the operations of the quantum computer and its states become entangled with the states of the environment. Decoherence tends to destroy irreversibly information in a superposition of states in a quantum computer in a way we cannot control and t herefore long computations, as well as long-term memories, seem to be impossible.

Successes of quantum information processing Perhaps the main apt killer for quantum information processing so far are Shor's quantum polynomial algorithms to factorize integers and to compute discrete logarithm and Grover's result that one can search in an unordered database of n elements using only $\mathcal{O}(\sqrt{n})$ queries. and not $\mathcal{O}(n)$ queries as in the classical case. The main open problem is whether there are quantum polynomial algorithms to compute the so-called Hidden subgroup problem for non-Abelian groups.

Quantum teleportation, due to Bennett and Brassard, experimentally verified, has been shown to be the most novel and useful tool for quantum communication.

Quantum cryptography, initiated by Wiesner, showed that provably, on the basis of the laws and limitations of quantum physics, unconditionally secure quantum key generation is possible. This brings new dimension to the problem of secure communication.

Due to the ideas coming from theory several ways have been found to fight quantum decoherence – the main enemy of the potential quantum computers – for example, Shor's quantum error-correcting codes and qauntum fault-tolerant computations.

Main quantum challenges can now be seen as follows.

- To understand quantum measurement and to get insights how much resources it needs.
- To study quantum entanglement, and its different types, as a new and powerful computation and communication resource – to study ways entanglement is detected, measured, created, stored, transmitted (even teleported), manipulated, transformed (from one form to another), quantified and consumed (to do some useful work).
- To study multipartite entanglement, its use for distributed quantum information processing and for physics in general and to discover the laws and limitations how entanglement can be shared.
- To study quantum (pure) states as a precious resources. (Of the large importance is to find out how much of quantumness we really need and how pure it has to be in order to have systems that are still more powerful than classical ones).
- To develop quantum information theory. The importance of quantum information theory follows from the observation that if we want to be able to understand and to utilize fully the information processing potential available in Nature, then the concepts of classical information theory need to be expanded to accumulate quantum information carriers.
- To discover new techniques (in addition to those of Simon/Shor and Grover) to design quantum algorithms more powerful than classical ones – if they exist.
- To deal with quantum mysteries. (The existence of various mysterious quantum phenomena, often termed as paradoxes, accompanies the development of quantum mechanics from its beginnings.) Quantum mysteries did not used to bather too much "working physicists" and therefore there has been a tendency to see these mysteries as topics mainly the philosophers of science should deal with. However, now when research in quantum information processing and communication aims to discover the laws and limitations of both physical and informational world, the existence of these mysteries cannot be ignored any longer. Moreover, theoretical informatics seems to have a potential to help to deal with these mysteries.

Conclusion Potential of QIPC starts to be widely recognized. A number of national and international programs and initiatives has been set up for QIPC. For basics and main outcomes of QIPC see Gruska (1999). For main challenges of QIPC, especially for theoretical informatics, see Gruska (1999a).

References

1. Jozef Gruska. Quantum challenges. In *Proceedings of SOFSEM'99, LNCS 1375*, pages 1–28, 1999.
2. Jozef Gruska. *Quantum computing*. McGraw-Hill, 1999. See also additions and updatings of the book on http://www.mcgraw-hill.co.uk/gruska.

Two Problems in Wide Area Network Programming*
(Position Statement)

Ugo Montanari

Dipartimento di Informatica, Università di Pisa, Italy
ugo@di.unipi.it

Motivations Highly distributed networks have now become a common platform for large scale distributed programming. Internet applications distinguish themselves from traditional applications on *scalability* (huge number of users and nodes), *connectivity* (both availability and bandwidth), *heterogeneity* (operating systems and application software) and *autonomy* (of administration domains having strong control of their resources). Hence, new programming paradigms (thin client and application servers, collaborative "peer-to-peer", code-on-demand, mobile agents) seem more appropriate for applications over internet.

These emerging programming paradigms require on the one hand mechanisms to support mobility of code and computations, and effective infrastructures to support coordination and control of dynamically loaded software modules. On the other hand, an abstract semantic framework to formalize the *model of computation* of internet applications is clearly needed. Such a semantic framework may provide the formal basis to discuss and motivate controversial design/implementation issues and to state and certify properties in a rigorous way.

Concern for the limited understanding we have of network infrastructure and its applications has been explicitly expressed in the US PITAC documents [9]. Also a theme on mobile/distributed reactive systems has been suggested as a new, proactive initiative for long-term, innovative research in a recent meeting promoted by the European Commission within the FET part of the V Framework programme.

Here we point out and shortly outline two issues which arise in the definition of such an abstract semantic framework.

Synchronization in a Coordination Framework Coordination [4] is a key concept for modeling and designing heterogeneous, distributed, open ended systems. It applies typically to systems consisting of a large number of software components, - considered as black boxes - which are independently programmed in different languages, and may change their configuration during execution.

While most of the activity is asynchronous, some applications, typically computer supported collaborative work or transactions among multiple partners, need primitives for synchronization. Synchronization might consist of complex computations to be performed by all partners on shared data before the global

* Research supported by TMR Network *G*ETGRATS and by MURST project *TOSCa*.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 609–611, 2000.
© Springer-Verlag Berlin Heidelberg 2000

commit action takes place. While these primitives will be implemented in terms of asynchronous protocols in the lower levels of system software, it is important to offer to the user a high level model of them, to be employed for specification, validation and verification.

The concepts developed for concurrent access to data bases, like serializability or nested transactions, are not fully adequate in this setting, since they refer to restricted models of computation. Instead, such issues should be considered like name and process mobility, causal dependencies between interactions, and refinement steps in the design process. We see this as a challenging problem, with aspects of logic, semantics, concurrency theory, programming languages, software architectures, constraint solving and distributed algorithms.

As a contribution we just mention two pieces of work by the author and collaborators. The first is about a generalized version of Petri nets: A *zero-safe net* [1,2] is in a *stable* state when certain places are empty. Step sequences from stable states to stable states through non-stable states are transactions, and correspond to transitions of an *abstract* net. The second piece of work is a model, called *tile logic* [7,3] (see http://www.di.unipi.it/~ugo/tiles.html) which extends structured operational semantics (SOS) by Gordon Plotkin and rewriting logic by Jose Meseguer. Tiles are inference rules which can be combined horizontally to build transactions and vertically to build ordinary computations.

Finite State Verification with Name Creation Finite state verification is possible when threads of control are independent of the actual data. In this case an automaton encompassing the whole state space can be constructed, possibly minimized, and checked for properties of interest expressed in some modal or temporal logic. This technique has been successfully and extensively applied to hardware and protocols.

When the computation involves the creation of new names which can occur in transition labels, the ordinary finite state techniques cannot be applied, since dynamic allocation of names with reuse of the old ones is required if the states must remain finite. On the other hand, in the coordination approach control is actually often independent from data, since computation mostly consists of redirecting streams, connecting and disconnecting users, transferring them from an ambient to another, resolving conflicts and performing security checks. Creation of new names is actually quite common in wide area programming, since nonces generated during secure sessions, process locations, and causes of forthcoming events can also be represented as names. Finite representation techniques for such systems together with expressive logical frameworks and efficient algorithms are needed for checking security and other global and local properties of distributed applications.

Some experience has been described in the literature with π-calculus verification [10,5,6]. Also Marco Pistore and the author have introduced certain classes of automata, called *History Dependent* (HD) [8], which are able to allocate and garbage collect names. Behavioral properties related to dynamic network connectivity, locality of resources and processes and causality among events can be formally verified on finite HD-automata. However efficiency is not satisfac-

tory and lots of work remain to be done about the theoretical and practical applicability of these methods.

References

1. R. Bruni and U. Montanari. Zero-safe nets: Comparing the Collective and Individual Token Approaches. *Inform. and Comput.*, 156:46–89. Academic Press, 2000.
2. R. Bruni and U. Montanari. Executing Transactions in Zero-Safe Nets. *Proc. International Conference on Petri Nets 2000*, Aarhus, to appear.
3. R. Bruni, U. Montanari and V. Sassone. Open Ended Systems, Dynamic Bisimulation and Tile Logic, to appear in Proc IFIP TCS2000, Sendai. *Proc IFIP TCS2000*, Sendai, this volume.
4. N. Carriero and D. Gelenter. Coordination Languages and Their Significance. *Communications of the ACM*, 35(2), 97–107, 1992.
5. M. Dam. Model Checking Mobile Processes. *Information and Computation 129(1)*, 1996, pp. 35-51.
6. G. Ferrari, S. Gnesi, U. Montanari, M. Pistore and G. Ristori. Verifying Mobile Processes in the HAL Environment In: Alan J. Hu and Moshe Y. Vardi, Eds., *CAV'98*, Springer LNCS 1427, pp.511-515.
7. F. Gadducci and U. Montanari. The Tile Model. In: G. Plotkin, C. Stirling and M. Tofte, Eds., *Proofs, Languages and Interaction: Essays in Honour of Robin Milner*, MIT Press, to appear.
8. U. Montanari and M. Pistore. An Introduction to History Dependent Automata. In: Andrew Gordon, Andrew Pitts and Carolyn Talcott, Eds, Second Workshop on Higher-Order Operational Techniques in Semantics (HOOTS II), ENTCS, Vol. 10, 1998.
9. President's Information Technology Advisory Committee. Information Technology Research: Investing in Our Future. Report to the President, National Coordination Office for Computing, Information, and Communications, February 1999, available from http://www.hpcc.gov/ac/report/.
10. B. Victor and F. Moller. The Mobility Workbench: A Tool for the π-calculus. *Proc. CAV'94*, Springer LNCS 818.

New Challenges for Computational Models
(Position Statement)

Yoshihito Toyama

Research Institute of Electrical Communication
Tohoku University
Katahira 2-1-1, Aoba-ku, Sendai 980-8577, Japan
toyama@nue.riec.tohoku.ac.jp

Theoretical computer science has offered various computation models like automata, Turing machine, lambda calculus, and their importance to theoretical work and to practice is widely recognized. The progress of the classical topics of formal language theory, computational complexity, and mathematical semantics of programming languages is due to these computation models. In this talk we propose two challenges that have tight connection to mathematical theory of programs and probably need new computation models to attack them.

Computation Model for Algebraic Programming

To set up algebraic equations is very common approach for many elementary mathematical problems, and we can easily find the solutions satisfying these equations by very simple algorismic method without deep knowledge of mathematics. The final goal of algebraic programming is to offer an analogous framework for deriving programs from algebraic specification for problems to that in elementary mathematics. Lambda calculus gives a useful mathematical basis in the study of lambda equations, but its theory is too mathematical and complicated for the practical use. Recent development of program transformation techniques by fusion and pairing shows an interesting direction of automatic algebraic derivation of functional programs, though their application is still too limited. The next step of algebraic programming seems to be to try to discover new computation models that unify the both directions.

Computation Model for Flexible Semantics

The most common mathematical presentations of operational semantics of programming languages are reduction systems. A reduction strategy chooses one out of many possible reductions, and can give an efficient deterministic interpreter through narrowing a given idealized nondeterministic semantics. The notion of reduction approximations is the inverse of reduction strategies, which broadens a given semantics to more flexible one. A decidable approximation, like the strongly sequential approximation, of term rewriting systems helps us to find a class of term rewriting systems having a decidable needed reduction strategy. Thus, not only narrowing but also broadening semantics is important to design an efficient deterministic interpreter for a given operational semantics.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 612–613, 2000.
© Springer-Verlag Berlin Heidelberg 2000

This observation easily leads us to an attractive idea of programming systems with flexible semantics, which can narrow or broaden the semantics of a programming language if necessary. Computation models for flexible semantics will provide new useful formal approaches to software science.

Towards a Computational Theory of Everything
(Position Statement)

Jiří Wiedermann*

Institute of Computer Science Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8
Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. Is there any common algorithmic principle behind the evolution of the universe, of life, of the mind? Should the currently prevailing computational paradigms which are limited by the Church–Turing thesis be revised in favor of more powerful ones? Can real systems exist whose computational power is provably greater than that of Turing machines? Is there a computational theory of everything? These are provocative questions that should be on the agenda of computer science for the next decade.

The universe gave birth to life, life developed the intelligence, and man constructed computers. Can this sequence be reversed? Can it be closed into an endless circle? Can a computer be intelligent? Can life emerge and exist within computers? Can man create a universe?

By the beginning of the 20th century it was not even possible to formulate questions as those above. Nowadays these questions, and similar ones, are becoming central topics in artificial intelligence and fundamental sciences such as physics, biology, mathematics, psychology, etc. In the intersection of the respective efforts there is the youngest of all fundamental sciences — computer science. At the present time, theoretical computer science is about to attack the respective problems by making use of its own tools and methods. Already now does it seem to possess the necessary knowledge and results that indicate at least partial answers to the questions at hand. The key to all answers lies in the very notion of computing.

What is computing? What entities can compute? How do we recognize that something computes, that something is 'computationally driven'? Where are the limits of computing?

Of course, computers can compute. Can people compute? Can the brain compute? There seems to be a tremendous difference between what brains and computers compute. Is this a principal difference caused by some so-far-unknown-to-us computational mechanism? Or is it merely a matter of efficiency? Or perhaps a matter of a different computational scenario? Or one of a different computational task that we cannot specify? Can the brain compute in a sense 'more'

* This research was supported by GA ČR Grant No. 201/00/1489

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 614–615, 2000.
© Springer-Verlag Berlin Heidelberg 2000

than computers? The brain can be conscious; it can think. Can a computer be conscious as well? Can it have emotions? Can it think? Is there a continuum of minds? What does the respective hierarchy look like? Is thinking an algorithmic process? Can we explain the emergence of mind, of the consciousness, of thinking, of language acquisition and generation in algorithmic terms? Can we understand understanding?

To the best of our current knowledge, the answer seems to be positive. The necessary ingredient in algorithmic modeling of the previously mentioned cognitive tasks includes learning via endless, and in a sense purposeful, interaction with the environment. What kind of mysterious self-organizing, self-checking learning algorithm lies behind the mind evolution? What is the corresponding computational model? Is it a mere neural net? A robotic body equipped with senses and driven by a neural net? Should we go to quantum phenomena when looking for an answer? What about biocomputers? Will they replicate themselves? Will they be alive then, in some sense? Will they be conscious? To what extent?

On a larger scale, one can ask: can nature compute? Does nature compute? Is evolution of life also a computational process? Where is its origin? What are the underlying algorithmic principles? Where are its computational limitations? Does the unpredictable, non-algorithmic nature of interaction among the evolutionary systems lead to surpassing the Church-Turing barrier? Again: what are the right computational models to capture the essence of the evolution? Are genetic algorithms the answer? Is there a single computational paradigm behind all that? Is the Internet an evolutionary system? Can we 'program' it in a way that will give rise to some artificial intelligence in it? Can an autonomous (software) agent emerge in the respective virtual environment and exist in it? Will such or similar form of virtual life keep developing ad infinitum? Will also the intelligence of such an agent grow above any limits?

Finally, on a still larger scale: the case of the Universe. Can we computationally model the genesis of the Universe? Out of what initial information? Are all these wonderful machines such as cellular automata, genetic algorithms, quantum computers, neural networks, biocomputers, internets, etc., indeed the right means to model what we are after? Are the known computational resources such as randomness, non-uniformity, fuzziness, quantum choice, interaction, really needed and/or sufficient to explain all phenomena of information exchange, forming and transformation? How faithful should our modeling be in order to capture all the necessary details? Where will the border be between the simulating system and the system to be simulated? Will then life emerge in our virtual universe? Will there eventually be some intelligence? And will it ask the same questions as we did at the beginning of our essay? Would its principal philosophical question be like this: "What was sooner: information, or the Universe?", or: "Is there a computational theory of everything?"

Today, at the doorstep to the third millennium, the issues mentioned above may sound to us as fantastically as would the questions from the beginning of this paper to men of science some 50 years ago. Or perhaps not?

On the Power of Interactive Computing*

Jan van Leeuwen[1] and Jiří Wiedermann[2]

[1] Department of Computer Science, Utrecht University,
Padualaan 14, 3584 CH Utrecht, the Netherlands.
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic.

Abstract. In a number of recent studies the question has arisen whether the familiar Church-Turing thesis is still adequate to capture the powers and limitations of modern computational systems. In this presentation we review two developments that may lead to an extension of the classical Turing machine paradigm: interactiveness, and global computing.

Computers will soon be part of all objects around us. It will enable forms of algorithmic intelligence and communication of unprecedented versality. The question arises whether the traditional notions in computability theory are still adequate to capture the powers and limitations of the new systems that are arising. In particular, some developments seem to challenge the view that computational systems are necessarily recursive. The present understandings of computation may transcend what is normally expressed by the Church-Turing thesis (see e.g. [4] and [12]). In this presentation we will explore some possible ingredients for extending the traditional computational models.

1 Interactive Computing

Among the powerful computational notions in the large and in the small, is the notion of 'interaction'. The notion typically applies to systems that consist of many components that compute and communicate with each other and with some external 'environment'.

The purpose of an interactive system is usually not to compute some final result but to react to or interact with the environment in which the system is placed and to maintain a well-defined action-reaction behavior. Interactive systems are always operating and thus may be seen as machines on infinite strings, but differ in the sense that their inputs are not specified and may depend on intermediate outputs and external sources.

In the late nineteen seventies and early nineteen eighties, interactive (or: reactive) systems received much attention in the theory of concurrent processes (see e.g. [7,8], [9]). Wegner [21,22] recently called for a more computational view

* This research was partially supported by GA ČR grant No. 201/98/0717 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 619–623, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of reactive systems, claiming that they have a richer behavior than 'algorithms' as we know them. He writes ([22], p. 318):

> *"The intuition that computing corresponds to formal computability by Turing machines ... breaks down when the notion of what is computable is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid."*

The study of 'machines' working on infinite input streams (ω-words) is by no means new and has a sizeable literature, with the first studies dating back to the nineteen sixties and seventies (cf. Thomas [15], Staiger [14]). Nevertheless the notion of interactiveness adds a new perspective to it. For a discussion of Wegner's claims from the viewpoint of computability theory, see Prasse and Rittgen [10]). Formalizations of the theory of interaction are studied in Wegner and Goldin [23,25], and Goldin [6].

In [18] we look at the computational implications of interaction. We give a simple model of interactive computing, consisting of one component C and an environment E interacting using single streams of input and output signals. The notion of 'component' that we use is very similar to that of a 'site machine' (see below). We identify a special condition, referred to as the interactiveness condition, which we impose throughout. Loosely speaking, the condition states that C is guaranteed to give some meaningful output within finite time any time after receiving a meaningful input from E and vice versa.

In the model we prove a number of general results for the interactive computing behaviour which a component C can exhibit, assuming that E can behave arbitrarily and unpredictably. We always assume that C is a program with unbounded memory, with a memory contents that is building up over time and that is never erased (unless the component explicitly does so). To understand the operation of interactive components, one may define a ω-string x to be interactively recognizable if C only outputs 1's (possibly interspersed with silent periods of finite duration) when it is fed x during its dialogue with E.

Example 1. The set $J = \{\alpha \in \{0,1\}^\omega | \alpha$ contains finitely many 1's$\}$ can be seen *not* to be interactively recognizable. Suppose there was an interactive component C that recognized J. Let E input 1's. By interactiveness C must generate a non-empty signal σ at some moment in time. E can now fool C as follows. If $\sigma = 0$, then let E switch to inputting 0's from this moment onward: it means that the resulting input belongs to J but C does not respond with all 1's. If $\sigma = 1$, then let E continue to input 1's. Possibly C outputs a few more 1's but there must come a moment that it outputs a 0. If it didn't then C would recognize the string $1^\omega \notin J$. As soon as C outputs a 0, let E switch to inputting 0's from this moment onward: it means that the resulting input still belongs to J but C does not recognize it properly. Contradiction.

Viewing components as interactive transducers of the signals that they receive from their environment one can show the following analogue to similar results

from classical automata theory, using suitable definitions. The result is quite involved and heavily relies on the interactiveness condition.

Theorem 1. *A set $J \subseteq \{0,1\}^\omega$ is interactively recognizable if and only if it can be interactively generated.*

In [18] we also study a notion of interactively computable functions. We prove that interactively computable functions are limit-continuous, using a suitable extension of the notion of continuity known from the semantics of computable functions. We also prove an interesting inversion theorem which states that interactively computable 1-1 functions have interactively computable inverses.

2 Global Computing

Among the new computational structures in the large, the Internet is beginning to play a very prominent role. Cardelli [3] has been among the first to realize the potential of the Internet as a programmable resource, suggesting that information structures may be realized on the Internet that can operate as 'global computers'. He foresees a multitude of (different) global computers operating on the Web, interacting with users and geographically distributed resources. The present developments in 'grid computing' (Foster and Kesselman [5]) are attempting to make this operational for the purposes of high performance computing.

Among the many questions raised by Cardelli [3], is the question what models of computation are appropriate for global computing. The Internet is an infrastructure for highly distributed forms of information exchange and computation, allowing large varying numbers of autonomous software entities to interact, compute and evolve under unpredictable circumstances. It is an environment in which the programs may depend on the data they have to operate on, the inputs are not given in advance and the computations may differ depending on the influence from unpredictable sources. This gives a computational power more akin to that of various 'non-uniform' computational models (cf. [1]), implying a power beyond that of ordinary Turing machines.

In [17] we describe a model of global computing in two stages. First 'site machines' are defined, by augmenting Turing machines with a communication facility. Next we define global Turing machines (or 'internet machines') as finite but time-varying sets of communicating site machines that can compute ad infinitum, modeling that a global computer may be evolving over time without limit. Under mild assumptions, the following theorem can be proved.

Theorem 2. *For every global Turing machine $\mathcal{M}$ there exists a Turing machine $\mathcal{T}$ with advice that sequentially realizes the same computations as $\mathcal{M}$ does, and vice versa.*

For an indication of the non-conventional power of Turing machines with advice we refer to Schöning [11] and Balcázar et al [1]. In [17] we also make a first step towards the development of a complexity theory for 'global space' and 'global time'.

3 Conclusion

Several developments in the theory of computation have challenged the kinds of computational models we are currently using (see [2], [13]). The given examples of interactive and global computing indicate that the classical Turing machine paradigm should be revised (extended) in order to capture the forms of computation that one observes in the systems and networks in modern information technology.

References

1. J.L. Balcázar, J. Diaz, and J. Gábarro. *Structural complexity*, Vol. I, 2nd edition, Springer-Verlag, Berlin, 1995.
2. L. Blum, F. Cucker, M. Shub and S. Smale. *Complexity of real computation*, Springer-Verlag, New York, NY, 1998.
3. L. Cardelli. Global computation, *ACM Sigplan Notices* 32-1 (1997) 66-68.
4. B.J. Copeland. The Church-Turing thesis, in: E.N. Zalta, *Stanford Encyclopedia of Philosophy*, Center for the Study of Language and Information (CSLI), Stanford University, Stanford, CA, http://plato.stanford.edu/entries/church-turing/, 1997.
5. I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*, Morgan-Kaufmann Publ., San Francisco, 1998
6. D.Q. Goldin. Persistent Turing machines as a model of interactive computation, in: K-D. Schewe and B. Thalheim (Eds.), *Foundations of Information and Knowledge Systems*, Proc. First Int. Symposium (FoIKS 2000), Lecture Notes in Computer Science, vol. 1762, Springer-Verlag, Berlin, 2000, pp. 116-135.
7. R. Milner. *A calculus of communicating systems*, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin, 1980.
8. R. Milner. Elements of interaction, *C.ACM* 36:1 (1993) 78-89.
9. A. Pnueli. Specification and development of reactive systems, in: H.-J. Kugler (Ed.), *Information Processing 86*, Proceedings IFIP 10th World Computer Congress, Elsevier Science Publishers (North-Holland), Amsterdam, 1986, pp. 845-858.
10. M. Prasse, P. Rittgen. Why Church's Thesis still holds. Some notes on Peter Wegner's tracts on interaction and computability, *The Computer Journal* 41 (1998) 357-362.
11. U. Schöning. Complexity theory and interaction, in: R. Herken (Ed.), *The Universal Turing Machine - A Half-Century Survey*, Oxford University Press, Oxford, 1988, pp. 561-580.
12. J,.C. Shepherdson. Mechanisms for computing over arbitrary structures, in: R. Herken (Ed.), *The Universal Turing Machine - A Half-Century Survey*, Oxford University Press, Oxford, 1988, pp. 581-601.
13. H.T. Siegelmann. Computations beyond the Turing limit, *Science* 268 (1995) 545-548.
14. L. Staiger. ω-Languages, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 6, Springer-Verlag, Berlin, 1997, pp. 339-387.
15. W. Thomas. Automata on infinite objects, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B: *Models and Semantics*, Elsevier Science Publishers, Amsterdam, 1990, pp. 135-191.

16. B.A. Trakhtenbrot. Automata and their interaction: definitional suggestions, in: G. Ciobanu and G. Păun (Eds.), *Fundamentals of Computation Theory*, Proc. 12th International Symposium (FCT'99), Lecture Notes in Computer Science, Vol. 1684, Springer-Verlag, Berlin, 1999, pp. 54-89.
17. J. van Leeuwen and J. Wiedermann. Breaking the Turing barrier: the case of the Internet, Techn. Rep. Institute of Computer Science, Academy of Sciences, Prague, 2000.
18. J. van Leeuwen and J. Wiedermann. A computational model of interaction, Techn. Rep. Dept. of Computer Science, Utrecht University, Utrecht, 2000.
19. P. Wegner. Interactive foundations of object-based programming, *IEEE Computer* 28:10 (1995) 70-72.
20. P. Wegner. Interaction as a basis for empirical computer science, *Comput. Surv.* 27 (1995) 45-48.
21. P. Wegner. Why interaction is more powerful than algorithms, *C.ACM* 40 (1997) 80-91.
22. P. Wegner. Interactive foundations of computing, *Theor. Comp. Sci.* 192 (1998) 315-351.
23. P. Wegner, D. Goldin. Coinductive models of finite computing agents, in: B. Jacobs and J. Rutten (Eds.), *CMCS'99-Coalgebraic Methods in Computer Science*, TCS: Electronic Notes in Theoretical Computer Science, Vol. 19, Elsevier, 1999.
24. P. Wegner, D. Goldin. Interaction as a framework for modeling, in: P. Chen *et al.* (Eds.), *Conceptual Modeling - Current Issues and Future Directions*, Lecture Notes in Computer Science, Vol. 1565, Springer-Verlag, Berlin, 1999, pp 243-257.
25. P. Wegner, D.Q. Goldin. Interaction, computability, and Church's thesis, *The Computer Journal* 1999 (to appear).

The Varieties of Programming Language Semantics (Summary)

Peter D. Mosses

BRICS & Dept. of Computer Science, Univ. of Aarhus, Denmark
http://www.brics.dk/~pdm/

Abstract. It may seem that there are almost as many varieties of programming language semantics as there are of programming languages. This brief summary surveys the main varieties of semantics, and considers which of them may be the most appropriate.

1 Introduction

In 1972, Christopher Strachey wrote the paper *The Varieties of Programming Language* [27]. (The title of the paper was by analogy with that of a book by William James: *The Varieties of Religious Experience* [10].) Strachey, in collaboration with Dana Scott, had developed the semantic description framework that became known as *denotational semantics*, and in the paper he showed how the relationship between the domains of so-called denotable, expressible, and storable values used in denotational semantics could reveal major characteristics of the described language.

In fact Strachey had an almost religious conviction in the superiority of the denotational approach to semantics over the operational and axiomatic approaches. He found it most natural to represent programming constructs as pure mathematical functions, and Scott-domains provided the foundations for his use of Curry's "paradoxical combinator" for obtaining fixed points of higher-order functions. Sadly, Strachey's work on the development of denotational semantics was cut short by his untimely death in 1975. His insight and ideas have nevertheless had a profound and lasting impact, as witnessed this year by the special issue of *HOSC* [2] published to commemorate the 25th anniversary of his death. The present author is particularly indebted to Strachey for the inspiration and guidance he provided during doctoral studies at the Programming Research Group in Oxford.

However, the proliferation of different frameworks for semantic description of programming languages over the past three decades makes one wonder whether Strachey's satisfaction with denotational semantics was fully justified. Let us (here, very briefly) survey the main varieties of semantics, and then consider which may be the most appropriate for various uses.

J. van Leeuwen et al. (Eds.): IFIP TCS 2000, LNCS 1872, pp. 624–628, 2000.
© Springer-Verlag Berlin Heidelberg 2000

2 Operational Semantics

The main idea of the operational approach to semantics is to model computations step-by-step. Operational frameworks include:

- translation to code for abstract machines such as the SECD-machine [12] or that of VDL [31];
- translation to the λ-calculus [13], which is then evaluated to normal form;
- application of abstract interpreters, which operate on a direct representation of the structure of the program [16];
- application of sets of term rewriting rules, using evaluation contexts [4] to control the order of sub-computations; and
- syntax-directed inference rules for transitions between configurations [1,11, 22,24], where "small-step" transitions model single steps of information processing, whereas "big-step" transitions model entire (terminating) computations, corresponding to the transitive closure of a small-step relation.

3 Denotational Semantics

Here semantic functions, generally defined inductively by semantic equations, map programs to their denotations, which represent computations as functions taking initial states directly to final states without revealing intermediate steps [7,19,26].

Variations on this theme include:

- initial algebra semantics [5], where (abstract) syntax is an initial algebra in a class of algebras, and where semantic equations are replaced by the definition of a target algebra interpreting the syntactic constructs;
- monadic semantics [14,17], where for each type of value, there is the type of computations of such values, and where the values and computations form a monad (equipped with further operations); and
- translation between meta-languages [18], where the semantic functions map programs to meta-language terms that may then be interpreted as functions in monads.

4 Axiomatic Semantics

The main idea here is to exploit assertions about programs:

- Hoare Logic [9] uses inference rules to relate assertions about the values of variables before and after program phrases;
- weakest-precondition semantics [3] is essentially denotational, with denotations being functions from assertions about values of variables after computations to assertions about the values beforehand; and
- laws of programming [25] characterize computation by assertions about program equivalence.

5 Hybrid Approaches

The final varieties of semantics considered here combine features of operational, denotational, and axiomatic semantics:

- action semantics [20,23,30] is similar to monadic denotational semantics, but denotations are actions rather than functions, and the notation used for composing computations is interpreted operationally;
- modular monadic action semantics [29] provides the action notation used in action semantics with a monadic denotational semantics;
- type-theoretic interpretation [8] uses evaluation contexts and reduction rules to define an internal language for composing computations;
- specification frameworks such as OBJ3 [6] and Rewriting Logic [15] can be used to define interpreters and transition relations; and
- mathematical operational semantics [28] allows definition by rules that provide both an operational and a denotational semantics.

6 Conclusion

To give a semantic description of a full high-level programming in *any* framework requires a major effort. Some of the varieties of semantics referenced above reduce the effort required by allowing re-use of parts of descriptions of other languages; these include modular SOS, monadic denotational semantics, action semantics, and modular monadic action semantics. Other important pragmatic aspects of semantic descriptions concern the ease with which they can be used for setting standards for implementations (or merely for communication of the designers' intentions to implementors), program verification, and compiler generation.

It appears that at present, no single semantic framework is ideal with regard to all pragmatic aspects. Sometimes, it is suggested that one should therefore aim provide two or more *complementary* semantic descriptions of each language, in different styles—and prove that they are equivalent—but the extra effort involved would surely be quite prohibitive when describing the semantics of larger programming languages.

In the opinion of this author, it is generally advantageous to translate a (large and complex) programming language into a (smaller and simpler) notation for semantic entities, as in denotational semantics, and in the hybrid action semantics and type-theoretic interpretation frameworks. However, when the distance between the programming language and the semantic notation is large, the translation itself may be uncomfortably complex; when it is small, the semantic notation may be almost as difficult to define or reason about as the programming language. The matter of how best to define the semantic notation itself appears to be less crucial, especially if the same semantic notation can be exploited in the description of many different programming languages.

Acknowledgements The author wishes to thank Professor Ito for encouragement to give an open lecture in conjunction with TCS2000 in Sendai, and for financial support. BRICS (Basic Research in Computer Science) is a centre of the Danish National Research Foundation.

References

1. E. Astesiano. Inductive and operational semantics. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Report, pages 51–136. Springer-Verlag, 1991.
2. O. Danvy and C. Talcott. Editorial. *Higher-Order and Symbolic Computation*, 13(1/2):5–6, 2000.
3. E. W. Dijkstra. Guarded commands, non-determinacy, and formal derivations of programs. *Commun. ACM*, 18:453–457, 1975.
4. M. Felleisen and D. P. Friedman. Control operators, the SECD machine, and the λ-calculus. In *Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986*, pages 193–217. North-Holland, 1987.
5. J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, 1977.
6. J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.
7. C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, editors, *Handbook of Theoretical Computer Science*, volume B, chapter 12. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
8. R. Harper and C. Stone. A type-theoretic interpretation of Standard ML. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Robin Milner Festschrifft*. MIT Press, 1998. To appear.
9. C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12:576–580, 1969.
10. W. James. *The Varieties of Religious Experience*. MacMillan Publishing Company, 1997. Reprint edition.
11. G. Kahn. Natural semantics. In *STACS'87, Proc. Symp. on Theoretical Aspects of Computer Science*, volume 247 of *LNCS*, pages 22–39. Springer-Verlag, 1987.
12. P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6:308–320, 1964.
13. P. J. Landin. A formal description of Algol60. In *Formal Language Description Languages for Computer Programming, Proc. IFIP TC2 Working Conference, 1964*, pages 266–294. IFIP, North-Holland, 1966.
14. S. Liang and P. Hudak. Modular denotational semantics for compiler construction. In *ESOP'96, Proc. 6th European Symposium on Programming, Linköping*, volume 1058 of *LNCS*, pages 219–234. Springer-Verlag, 1996.
15. N. Martí-Oliet and J. Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay, editor, *Handbook of Philosophical Logic*, volume 6. Kluwer Academic Publishers, 1998. Also Technical Report SRI-CSL-93-05, SRI International, August 1993.
16. J. McCarthy. Towards a mathematical science of computation. In *Information Processing 62, Proc. IFIP Congress 62*, pages 21–28. North-Holland, 1962.

17. E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Computer Science Dept., University of Edinburgh, 1990.
18. E. Moggi. Metalanguages and applications. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute. CUP, 1997.
19. P. D. Mosses. Denotational semantics. In *Handbook of Theoretical Computer Science*, volume B, chapter 11. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
20. P. D. Mosses. *Action Semantics*. Number 26 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1992.
21. P. D. Mosses. Foundations of modular SOS. Research Series BRICS-RS-99-54, BRICS, Dept. of Computer Science, Univ. of Aarhus, 1999. Full version of [22].
22. P. D. Mosses. Foundations of Modular SOS (extended abstract). In *MFCS'99, Proc. 24th Intl. Symp. on Mathematical Foundations of Computer Science, Szklarska-Poreba, Poland*, volume 1672 of *LNCS*, pages 70–80. Springer-Verlag, 1999. Full version available [21].
23. P. D. Mosses and D. A. Watt. The use of action semantics. In *Formal Description of Programming Concepts III, Proc. IFIP TC2 Working Conference, Gl. Avernæs, 1986*, pages 135–166. North-Holland, 1987.
24. G. D. Plotkin. A structural approach to operational semantics. Lecture Notes DAIMI FN–19, Dept. of Computer Science, Univ. of Aarhus, 1981.
25. A. W. Roscoe and C. A. R. Hoare. The laws of occam programming. Technical Report PRG-53, Programming Research Group, Oxford University, 1986.
26. D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In *Proc. Symp. on Computers and Automata*, volume 21 of *Microwave Research Institute Symposia Series*. Polytechnic Institute of Brooklyn, 1971.
27. C. Strachey. The varieties of programming language. In *Proc. International Computing Symposium*, pages 222–233. Cini Foundation, Venice, 1972. A revised and slightly expanded version is Tech. Mono. PRG–10, Programming Research Group, University of Oxford, 1973.
28. D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS'97*. IEEE, 1997.
29. K. Wansbrough and J. Hamer. A modular monadic action semantics. In *Conference on Domain-Specific Languages*, pages 157–170. The USENIX Association, 1997.
30. D. A. Watt. *Programming Language Syntax and Semantics*. Prentice-Hall, 1991.
31. P. Wegner. The Vienna definition language. *ACM Comput. Surv.*, 4:5–63, 1972.

Author Index

Abadi, Martín, 3
Albrecht, Andreas, 301
Ausiello, Giorgio, 602

Blundo, Carlo, 140
Boros, Endre, 257
Brimkov, Valentin E., 286
Bruni, Roberto, 440
Buchholz, Thomas, 213

Cardelli, Luca, 333
Chandy, K. Mani, 580
Charpentier, Michel, 580
Chen, Zhi-Zhong, 84
Cho, Sung-Up, 76
Choi, Mi-Young, 76
Chung, Yoojin, 100

Dal Zilio, Silvano, 409
Dantchev, Stefan S., 286
Despeyroux, Joëlle, 425

Eidenbenz, Stephan, 200

Fiore, Marcelo P., 457
Flajolet, Philippe, 152
Fournet, Cédric, 348
Fredriksson, Kimmo, 59
Fujita, Ken-Etsu, 505

Galdi, Clemente, 140
Ghelli, Giorgio, 333
Gordon, Andrew D., 333
Gruska, Jozef, 599, 604

Hagiya, Masami, 23
Hatzis, Kostas, 152
Henzinger, Thomas A., 549

Ibaraki, Toshihide, 257

Jeong, Chang-Sung, 76

Kameyama, Yukiyoshi, 489
Klein, Andreas, 213
Kleist, Josva, 390
Kobayashi, Naoki, 365
Kutrib, Martin, 213
Kwon, Hyuk-Chul, 100

Leeuwen, Jan van, 619
Lermer, Karl, 564
Lévy, Jean-Jacques, 348
Löding, Christof, 521
Loyer, Yann, 536

Makino, Kazuhisa, 257
Martin, Bruno, 226
Mauri, Giancarlo, 45
Mayr, Ernst W., 99
Mayr, Richard, 474
Merro, Massimo, 390
Mishra Sounaka, 186
Mizuki, Takaaki, 273
Montanari, Ugo, 440, 609
Mosses, Peter D., 624

Navarro, Gonzalo, 59
Nestmann, Uwe, 390
Nikoletseas, Sotiris, 152
Nishizeki, Takao, 273

Park, Kunsoo, 100
Pavesi, Giulio, 45
Plotkin, Gordon D., 596

Rogaway, Phillip, 3

Sarzeaud, Olivier, 126
Sassone, Vladimiro, 440
Schmitt Alan, 348
Schubert, Aleksy, 505
Shchepin, Evgeny V., 112
Shizuya, Hiroki, 273
Sikdar Kripasindhu, 186
Spirakis, Paul, 152
Spyratos, Nicolas, 536
Stéphan, Yann, 126
Stamate, Daniel, 536
Stamm, Christoph, 200

Strooper, Paul, 564
Sudan, Madhu, 25

Taoka, Satoshi, 169
Tezuka, Shu, 243
Thomas, Wolfgang, 521
Toyama, Yoshihito, 612

Uehara, Ryuhei, 84
Ukkonen, Esko, 59

Vakhania, Nodari N., 112

Watanabe, Toshimasa, 169
Whang, Sun-Chul, 76
Wiedermann, Jiří, 614, 619
Wong, Chak-Kuen, 301

Yannakakis, Mihalis, 315